VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Rámec pro geolokační hry

Framework for Geolocation Games

2018 Lucie Kuchárová

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Lucie Kuchárová**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Rámec pro geolokační hry
Framework for Geolocation Games

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem práce je vytvořit rámec pro geolokační hry, který umožní vytvářet různé verze her založených na určování polohy hráče a jeho navigace k zájmovým bodům hry, kde budou řešeny různé úkoly.
Implementace rámce umožní:
1. Vytváření a editaci geolokačních her.
2. Správu uživatelů a her.
3. Rozšiřování jednotlivých typů úkolů.

Práce bude obsahovat:
1. Průzkum aktuálního stavu geolokačních her.
2. Analýza a návrh geolokačního rámce.
3. Prozkoumání možnosti offline režimu.
4. Implementaci rámce a ukázkové použití.
5. Zhodnocení rámce z hlediska výkonu a přeneseného množství dat, zatížení mobilního zařízení.

Seznam doporučené odborné literatury:

[1] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides (Gang of Four): Návrh programů pomocí vzorů. Grada. Praha 2003. ISBN 8024703025
[2] DARWIN, Ian F. Java cookbook. 2nd ed. Sebastopol, CA: O'Reilly, c2004, xxiv, 829 p. ISBN 05-960-0701-9. Dostupné z: http://it-ebooks.info/book/2249/

Dále dle pokynů vedoucího práce.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. David Ježek, Ph.D.**

Datum zadání: 01.09.2017

Datum odevzdání: 30.04.2018

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracovala samostatně. Uvedla jsem všechny literární prameny a publikace, ze kterých jsem čerpala.

V Ostravě 30. dubna 2018 .....

Chtěla bych poděkovat svému vedoucímu práce Ing. Davidu Ježkovi Ph.D. za rady, pomoc a vstřícnost při konzultacích, a za odborné vedení diplomové práce.

Abstrakt

Tato dimplomová práce se zabývá návrhem a zhotovením geolokačního rámce, který umožní vytvářet různé verze her založených na určování polohy hráče. Práce analyzuje stávající možnosti her na mobilních zařízeních, jejich funkcionalitu i problémy. Z toho čerpá ve vlastním návrhu rámce, který problémům, v již existujících hrách, předchází a poskytuje webové rozhraní pro vytváření her vlastních. Tento rámec pak poskytuje jednoduchý způsob, kterým je možné stávající funkcionalitu rozšířit pro vznik nových typů herních akcí.

Klíčová slova: GPS, geolokace, mobilní aplikace, Android, datová spotřeba, synchronizace dat, offline režim, SOA, mikroslužby, zprostředkovatel, událostmi řízená architektura, Open Street Map, rozšiřitelnost, AngularJS, Kafka, WebSocket, JWT, Docker

Abstract

This thesis deals with design and implementation of geolocation framework. This framework enables creation of different game versions based on defining current position of player. This work analyses current game posibilities on mobile devices, their functionalities and problems. This information is used for designing geolocation framework with focus on elimination of these problems, offering creation of own games via web editor. This framework also offers simple way to extend existing functionalities and allows implementation of new game actions.

Key Words: GPS, geolocation, mobile device, Android, data usage, data synchronization, offline mode, SOA, microservices, broker, event driven architecture, Open Street Map, extensibility, AngularJS, Kafka, Websocker, JWT, Docker

Obsah

Seznam použitých zkratek a symbolů

GPS	–	Global positioning system
API	–	Application programming interface
GC	–	Geocaching
USA	–	United States of America
MMO	–	Massively multiplayer online (game)
MMORPG	–	Massively multiplayer online role-playing game
EU	–	Evropská unie
Wi-Fi	–	Bezdrátová komunikace v počítačových sítích
ID	–	Unikátní identifikátor
SOA	–	Service oriented architecture
ESB	–	Enterprise service bus
UI	–	Uživatelské rozhraní
URL	–	Uniform Resource Locator
MVC	–	Model-View-Controller
HTML	–	HyperText Markup Language
JPA	–	Java Persistence API
TCP	–	Transmission Control Protocol
JPA	–	Java Persistence API
WS	–	WebSocket
JWT	–	Json Web Token
AWS	–	Amazon Web Services
GB	–	Gigabyte
RAM	–	Random access memory

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

Geolokační hry jsou s námi už od nepaměti. Určitě každý z nás se už někdy účastnil honby za pokladem, viděl poletující fáborky nebo nakreslené šipky v parku vedoucí zdánlivě nikam, nebo alespoň slyšel o tajemných výpravách, u kterých si nikdy nejste jisti, kam vás dovedou, nebo co na konci můžete očekávat. Vyluštění jednoho vodítka nás navede k místu, kde hledáme vodítko další, hra nás provází různými oblastmi, a když se zadaří, na konci čeká odměna. Někdy ale taková hra nemusí být čistě pro zábavu, pro některé z nás může být takovou geolokační hrou cesta v cizím městě z bodu A do bodu B, aniž bychom se ztratili. Ať už jsme ale lovci pokladů nebo ztracení turisté, potřebujeme nějakou formu navigace.

Kdo si už někdy dal práci s přivazováním zmíněných fáborků, schováváním papírků s vodítky a kreslením křídou po cestách, určitě narazil na problém. Když počasí nepřeje, vítr fouká moc silně nebo na chvíli zaprší, naše přípravy přijdou vniveč a zbude nám pouze lítost nad ztraceným časem a zklamání nad zábavou, která neproběhne. Kdo už se někdy chystal objevit krásy cizích míst, hledal zajímavosti a zaznačoval místa, která navštívit, mohl zadoufat, že místo plánování hned vyjede a na místě potká ochotného místního průvodce, který by přesně věděl, co by ho zajímalo.

Velký rozvoj nových technologií a chytrých mobilních zařízení dokáže takové nepříjemnosti bez problému vyhladit. Z nepřeberého množství aplikací si můžeme vybrat právě tu pro nás. Právě geolokační hry jsou v posledních letech čím dál víc oblíbenější formou zábavy. Technologie, které jsou dnes v našem světě dostupné téměř každému, spojují se zábavou, fyzickou aktivitou, někdy i lidskou potřebou socializace i osvětou.

Co když ale nenajdeme právě tu aplikaci, jakou bychom chtěli? Řekněme, že za účelem tématické párty chceme vytvořit dobrodružnou cestu středověkem napadeném mimozemšťany v místě našeho bydliště. Šance, že existuje mobilní aplikace, která nás takovým příběhem provede, je mizivá.

Vize, která mě vedla k práci na tomto tématu, řeší tento problém. Obecně řečeno, geolokační hry potřebují ke své existenci několik věcí - rozhraní s mapou, které je pro hráče "vstupní branou" do světa rozšířené reality, GPS připojení pro zaměření lokace hráče a případně aktivní internetové připojení. Smyslem této diplomové práce je vytvořit geolokační rámec tak, aby poskytl funkční základ pro spojení vyjmenovaných prvků a zároveň umožnil uživateli být aktivní nejen z pohledu hráče, ale také z pohledu vytvoření samotného smyslu a příběhu hry. Uživatel bude mít při interakci s výsledným systémem k dispozici editor, ve kterém bude moci vytvořit vlastní body zájmu i herní prvky a vytvořený svět bude mít možnost zpřístupnit ostatním hráčům přes mobilní aplikaci. Důraz bude kladen zejména na modularitu řešení, aby různé typy úkolů a objektů byly snadno zaměnitelné a rozšiřitelné a výsledná hra mohla být poskládána z mnoha odlišných modulů, umožnujících uživateli co největší diverzitu v možnostech sestavení hry. Zároveň se práce zaměří na problémy, kterým dnešní geolokační hry čelí a zohlední je ve vlastní implementaci.

Pro zmíněnou tématickou párty by bylo řešením využití speciálního editoru, ve kterém uživatel zadá hlavní dějovou linii hry, vytvoří například objekt hostince, zbrojnice a nepřátelské vesmírné lodi, každému z těchto objektů přiřadí určité vlastnosti (v hostinci hráč nabírá sílu, u vesmírné lodi je pod palbou laserových zbraní a podobně), umístí je na mapu a potvrdí vytvoření světa. Při spuštění mobilní aplikace bude moci svůj svět najít, spustit a při procházení okolí vidět objekty na mapě tam, kde je v editoru umístil.

Samotnému návrhu a implementaci geolokačního rámce v této práci předchází analýza již stávajících geolokačních her. Některé konkrétní hry popíši o něco více a zaměřím se i na problémy, o kterých se ví, že jim tvůrci těchto her čelili.

Následovat bude návrh rámce, kde se budu snažit nastínit možné principy a vlastnosti rámce, primárně z hlediska koncového uživatele. Budou uvedeny příklady různých částí systému a funkcionalit, které by mohly nabízet.

V další části se zaměřím na sestavení architektury. Zde nejprve uvedu obecné principy a vzory, na kterých se bude výsledná architektura zakládat. Poté představím mnou navrženou architekturu a uvedu, jaké zodpovědnosti budou mít jednotlivé části systému.

Následující část se bude věnovat konkrétním implementačním principům, postupům a technologiím, kterých bylo při ukázkové implementaci rámce využito. Součástí popisu implementovaných komponent je pak představení postupu, kterým je zajištěna rozšířitelnost systému o nový typ úkolu.

Dalším prvkem v této diplomové práci je představení řešení, které se stará o správu komponent z hlediska jejich sestavení a konečné spustitelnosti.

Poslední kapitola pak pojednává o kladech a záporech použitých technik v souvislosti s výkonem systému a přenášených dat mezi systémem a uživatelem.

2 Stávající geolokační hry

Jak jsem již zmínila v úvodu, s větší dostupností potřebných geolokačních technologií se rozmohlo i povědomí o geolokačních hrách. V následujících řádcích tedy zmíním pár z těch nejrozšířenějších, popíšu, jaké zajímavé možnosti přinášejí, na jakých principech fungují a u některých nastíním i problémy, kterým musely, nebo stále musejí, čelit. Hry jsem rozdělila do tří pomyslných částí. První částí jsou hry, které jsou ve své podstatě založeny pouze na souřadnicích GPS. Do další části jsem zařadila hry, které GPS využívají také, ale nabízejí ještě něco navíc a poskytují něco málo i z rozšířené reality. V poslední části se budu věnovat hrám, které využívají možnosti prvních dvou částí, herní akce už se ale dějí v reálném čase a je proto třeba mít stálé internetové připojení.

2.1 GPS

Základem pro určování polohy kdekoli ve světě je bezpochyby Globální polohový systém - GPS. Tato technologie se rozvíjela od druhé poloviny 20. století a je provozována ministerstvem obrany Spojených států amerických. Její vznik byl, jako mnoho jiných technologií, zapříčiněn primárně z vojenských důvodů, ve svých počátcích tedy nebyla přístupná veřejnosti. Z důvodu bezpečnosti ministerstvo obrany USA záměrně rušilo GPS signál až do roku 2000, kdy prezident Bill Clinton nařídil ukončení umělé degradace přesnosti veřejného GPS signálu a technologii zpřístupnil civilnímu obyvatelstvu[17].

2.1.1 Geocaching

Světoznámá hra Geocaching, založená na hledání keší (reálných "pokladů" umístěných do skrýší na nejrůznějších místech světa), se od zmíněného roku 2000, kdy byla hra zahájena založením prvotní keše, vyšplhala na přibližně 7 milionů aktivních hráčů a 3 miliony ukrytých keší[18]. Keše jsou malé krabičky, boxy nebo schránky s listem pro podpisy hráčů, kteří ji najdou, zakládány samotnými hráči na historických, vyhlídkových či jinak zajímavých místech, o která se zakladatelé chtějí podělit s ostatními hráči. "Pokladem" tedy ve skutečnosti není samotná keš, ale místo, které reprezentuje. Díky tomu se hráči dostávají na pozoruhodná místa, na která by je někdy ani nenapadlo jít, přičemž se často setkávají s ostatními hráči a spojují své síly při hledání.

Pro hraní této hry není zapotřebí nic jiného, než znalost GPS souřadnic dané keše. Žádné mobilní aplikace nejsou třeba, rozmach dostupnosti chytrých telefonů a tabletů však hráčům značně pomohl. Díky nim je hledání keší mnohem snadnější, stačí si nainstalovat jednu z určených aplikací (například Geocaching nebo c:geo[16]) a mapu se zaznačenými kešemi tak mít stále u sebe. Aplikace určené pro Geocaching využívají API, které umožňuje přístup k databázi všech herních objektů, a zprostředkovávají tak informace nejen o poloze keší, ale také o poloze uživatele a pomocí GPS souřadnic a kompasu jsou schopny jej k daným keším navést. Softwarové rozhraní

toto umožňující bývalo veřejné, avšak před několika lety se provozovatelé rozhodli omezit k API přístup a nové aplikace bohužel nevznikají. Uživatelé však mají na výběr z aplikací stávajících, které jsou stále podporovány, a vývojáři mohou využít některou z aplikací třetích stran, které mají stále k oficiálnímu API přístup a zprostředkovávají jej dál, kupříkladu GC Live API[15].

2.2 GPS + aplikace

Kromě samotného hledání schovaného pokladu se otevřely dveře i dalším možnostem, jak pojmout koncept geolokačních her, a vznikly aplikace přímo určené pro použití s mobilními zařízeními. Zařadila bych zde aplikace typu průvodce, které nám kromě bodů na mapě mohou předávat i mnoho informací o místech, která procházíme, ať už je to jejich historie nebo hodnocení lokálních restaurací.

2.2.1 Kód Salomon

Jednou takovou aplikací je Kód Salomon[19], hra, která má za cíl provést člověka přes zajímavé lokace v Ostravě, procvičit jeho mozkové buňky řešením šifer a předat informace o historii Ostravy. Hru je možné si zahrát s více scénáři, přičemž každý z nich uživatele zavede na jiná místa a předá mu různé informace. Součástí hry jsou i videa, což se může nepříjemně projevit na množství přenášených dat. Hra má však možnost offline režimu, kdy je možné si videa a jiný multimediální obsah předem stáhnout a uložit v mobilním zařízení, je tedy možné hrát pouze s aktivním připojením GPS.

2.2.2 Zombies, Run!

Také aplikace sledující sportovní výkony jako Runkeeper, Sports Tracker nebo Endomondo jsou velmi využívanými geolokačními aplikacemi. Toho využila hra s názvem Zombies, Run!, která se zabývá jak monitorováním běhu, tak provázením běžce příběhem, v němž je zasazen do postapokalyptického světa a je upozorňován na neustálé nebezpečí blížících se Zombies, před kterými musí utíkat. Hra se doporučuje hrát se sluchátky, hráč při běhu virtuálně plní "mise", při kterých mu jsou předávány informace ve formě simulovaného rádiového spojení. Jako u předchozí hry, i zde je, z možného důvodu velké datové náročnosti, možnost stažení dat jednotlivých misí předem.

2.3 GPS + aplikace + internetové připojení

Jiný typ geolokačních her už vyžaduje neustálé internetové připojení, jsou to například hry označované jako MMO Games či MMORPG. V poslední době proslavený Pokémon GO a Ingress od společnosti Niantic nebo Resources vývojáře UN3X potřebuje nepřetržité aktualizace stavu světa pro všechny připojené hráče.

Ingress je založen na boji dvou frakcí, dobývání portálů umístěných na reálných místech na mapě a obsazování co největší části světa danou frakcí. V průběhu hry hráč sbírá XP hmotu

soustředěnou okolo portálů, kterou následně využívá pro dobývání a různá vylepšení ve hře. Pokémon GO se třemi frakcemi, postaven na základech hry Ingress, má za cíl podobné obsazování "portálů" nazvaných gym, které provádí prostřednictvím chycených Pokémonů. Samotní hráči také mohou navrhnout založení nového portálu. Hra Resources je podobně založena na obsazování území, tentokrát ale hraje každý hráč sám za sebe, hledá ložiska surovin ve svém okolí, staví doly nebo napadá doly ostatních hráčů. U všech těchto her je důležité mít neustále ty nejaktuálnější data o hře, aby se celkový stav hry pro žádného hráče nelišil.

Autoři her Ingress a Pokémon GO se netají technologiemi, které jsou pro jejich provoz použity. Základem jejich úspěchu je sestavení takové architektury, která je provozuschopná na některé ze služeb obecně nazývaných *Cloud*. To jim zajišťuje velkou flexibilitu při velkém zatížení systému, kdy se připojí mnoho uživatelů a komponenty, které jsou tímto nejvíce vytěžovány, se automaticky replikují, aby pokryly požadované dotazy.

2.4 Problémy a možná řešení

2.4.1 Mobilní data

Prvním velmi důležitým problémem, který je nutno vzít v potaz, je množství přenášených dat, které aplikace přijímá nebo zasílá. Toto nemusí trápit uživatele s neomezeným datovým připojením, avšak mnoho uživatelů si objem přenášených dat hlídá a aplikace, které data využívají nadměrně mnoho, nepoužívá. Důležitým faktorem je zde finanční stránka, kdy jsou mobilní data jednoduše stále dost drahá. Kupříkladu, Česká republika má dle průzkumu Evropské unie jedny z nejdražších mobilních připojení v EU[14]

Řešení se zde nabízí v podobě možného stažení a uložení herních dat ještě před samotným hraním, aplikace se tak "v akci"oprostí od nadměrného toku dat a mobilní data pak využívá minimálně, jen pro akce, které jsou nezbytně nutné.

2.4.2 Výdrž baterie

Další nepříjemností, která také souvisí s problémem mobilních dat, je výdrž baterie mobilního zařízení. Čím víc data proudí, tím víc je naše zařízení zatěžováno a rychleji se vybíjí.

Největší dopad na vybíjení baterie mobilního zařízení má velmi často na svědomí aktivita displeje. Při hraní se uživatel zapnutému displeji samozřejmě nevyhne, je však možné do aplikace zakomponovat techniky, které budou schopny uživatele upozornit na událost ve hře, aniž by musel kontrolovat stav hry pohledem.

Vibracemi nebo zvukovou notifikací je možné dát uživateli najevo, že se, například, blíží k objektu zájmu, a tím tak snížit nutnost zapínání obrazovky, a tedy i spotřebu baterie. K tomu je potřeba, aby byl povolen běh aplikace na pozadí i po zhasnutí displeje.

Na vině velké spotřeby baterie může být, bohužel, pro tuto práci nepostradatelná, polohová služba GPS. Toto může sám uživatel ovlivnit, například v zařízeních se systémem Android je možné zvolit režim, ve kterém se bude aktuální lokace zjišťovat.

Nejméně náročný je režim, který využívá k nalezení polohy čistě modul GPS. Tento režim je však také nejméně přesný. Druhým stupněm je pak použití Wi-Fi a mobilních sítí a třetím využití všech tří služeb dohromady, který je ze všech tří nejnáročnější, zato ale nejvíce přesný.

2.4.3 Synchronizace dat

Synchronizace dat je problémem hlavně masivně využívaných her, ke kterým je v jednu chvíli připojeno velké množství uživatelů. Synchronizace dat tak musí probíhat neustále a v nejkratším možném čase.

S množstvím dat, které se v takovýchto aplikacích přelévá, je nutno počítat při návrhu komunikace v celkovém systému. Případná škálovatelnost a replikace zatížených komponent musí korespondovat s komunikačním kanálem a tento kanál se musí umět případnému připojení nové služby přizpůsobit.

3 Analýza a návrh geolokačního rámce

Jak již bylo nastíněno, hlavním požadavkem na výsledek této diplomové práce je vytvoření rámce, který by uživatelům nabídl šanci navrhovat vlastní verze her, které budou založeny na určování polohy hráčů prostřednictvím mobilních zařízení. Uživatelé by samozřejmě měli mít možnost k navržené hře přes mobilní aplikaci přistoupit a hrát.

Návrh rámce se ponese v souladu s předchozími kapitolami a bude se snažit o to, aby se problémům v nich zmíněným předešlo.

3.1 Interakce uživatelů se systémem

Rámec uživatelům nabídne dvě formy uživatelského rozhraní. První formou je webový editor, za pomocí kterého mohou hry, herní objekty i herní akce vznikat. Druhou formou je mobilní aplikace, která je schopna již vytvořené hry spouštět. Uživatelé musí mít definovány role, kterými se prokazují, a které zajišťují, že každý uživatel je schopen provádět jen ty akce, na které má v rámci hry nebo celého systému nárok. Aby mohl uživatel systém využívat, musí se nejprve registrovat. Vstupním prvkem do systému je vždy přihlášení. Detaily uživatelských rolí budou popsány v dalších kapitolách.

3.1.1 Webový editor

Webový editor je určen zejména pro uživatele, kteří chtějí být buď vlastníky, nebo spoluautory určité hry. Vlastník je člověk, který se rozhodne pro vytvoření nové hry. Součástí editoru musí být možnost vytvoření, úpravy a mazání her.

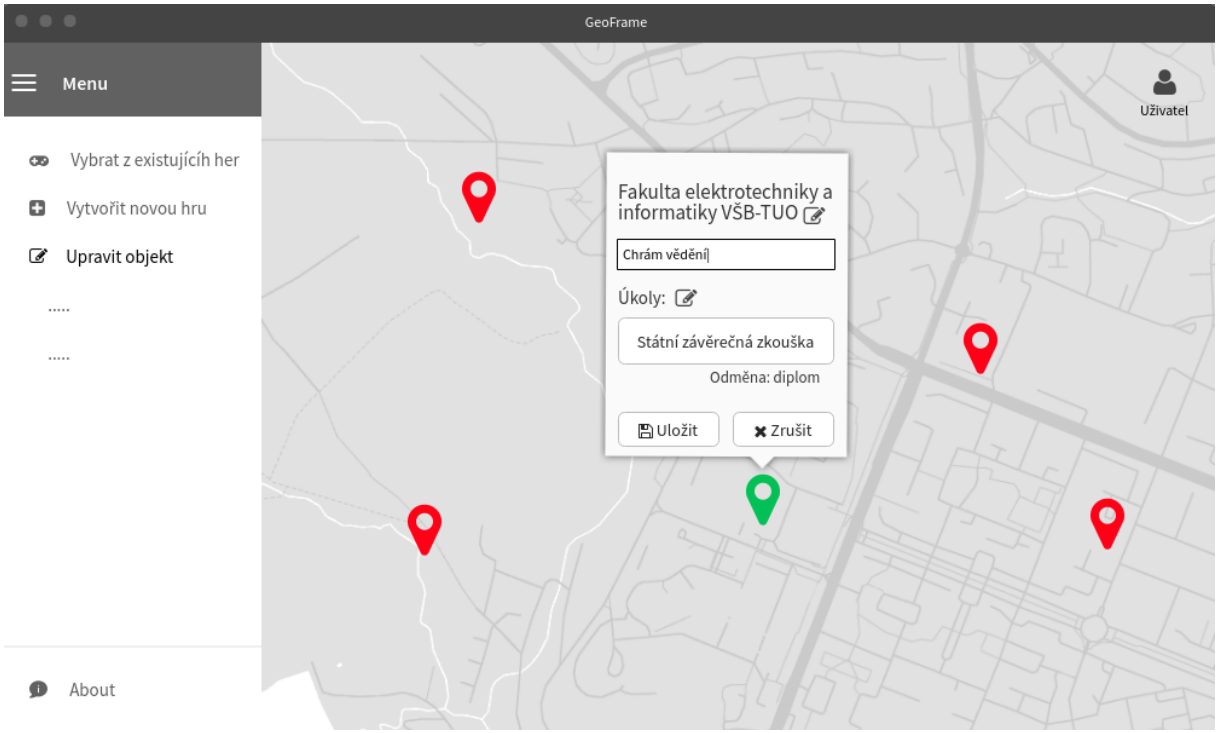
Vlastník dané hry má v této hře největší práva. Při vytváření hry zadá její název, popis, rozsah načítaných objektů, příběh a další vlastnosti, které budou pro hru nezbytné. Také má zde možnost zvolit editory, kteří takto získají práva k úpravě objektů ve hře.

Společně s editory pak mohou vlastníci vytvářet, upravovat a mazat objekty ve hře. Každý objekt má základní vlastnosti, které musí být specifikovány, jako je například název a jeho popis. Webový editor tedy musí mít prostředky k tomu, aby se s objekty dalo manipulovat.

Dále se ke každému objektu může vztahovat jakýkoli úkol. Ve webovém editoru musí být možnost nového úkolu vytvořit a napojit ho na existující objekt a možnost stávajícího úkolu upravit nebo odstranit.

3.1.2 Uživatelské rozhraní - webový editor

Hlavním prvkem geolokační hry je bezpochyby mapa s objekty, měla by být tedy očividným středem pozornosti. Zde je vhodné čerpat inspiraci z již existujících webových rozhraní pro mapy, například *google.com* nebo *mapy.cz*, které drtivá většina uživatelů zná a je zvyklá na to, jak fungují.



Obrázek 1: Návrh rozhraní webového editoru

Důležitá je jednoduchost používaného systému. Mělo by být na první pohled jasné, kde je třeba kliknout pro vytvoření nebo úpravu nové hry, jak přidat nový objekt nebo upravit stávající data. S objekty se bude pracovat interaktivně přímo v mapě. Například kliknutí na ikonu objektu v mapě, znázorňující jeho polohu, zobrazí základní informace o objektu a akce, které lze s objektem provést. Těmito akcemi se pak může uživatel dostat k podrobnějším informacím o objektu formou vyskakovacích oken, vytvářet nové skutečnosti a provádět složitější operace.

Návrh, jak by mohl editor vypadat, si lze prohlédnout na obrázku 1 na straně 20.

3.1.3 Mobilní aplikace

Mobilní aplikace je určena pro hráče. Po spuštění aplikace budou mít hráči možnost přihlásit se ke svému účtu a posléze vybrat hru, ke které se chtějí připojit. Pokud již uživatel danou hru hrál, po připojení bude mít příležitost pokračovat tam, kde skončil. Hra tedy musí zajistit, aby se ukládal stav hry pro jednotlivé uživatele.

Hráči, po připojení k určité hře a případnému načtení jejího stavu, se zobrazí mapa s jeho aktuální polohou. Načtou se i herní objekty v jeho bezprostředním okolí.

Velikost plochy, která se bude při přihlášení a pohybu načítat, bude specifikována danou hrou, popřípadě samotným uživatelem. Uživatel však svým nastavením nemůže zvětšit načítanou plochu nad rámec specifikace hry. Například, dovolí-li nastavení hry načítání objektů 500 metrů

od hráče, může si uživatel změnit nastavení na plochu menší, kupříkladu 100 metrů, ale větší, kupříkladu 600 metrů, už ne.

Toto nastavení bude možné v oddělené části uživatelského rozhraní, které bude součástí aplikace a bude obsahovat všechny preference konkrétního uživatele. V tomto nastavení bude také možné spustit *offline* režim hry, při kterém se zakáže komunikace aplikace s internetem. Toto nastavení bude závislé na povaze dané hry a na jejích možnostech. Hra pro více hráčů, postavená na principu interakce v reálném čase, kdy se uživatelé navzájem ovlivňují, nebude v tomto módu hratelná. Na rozdíl od tohoto hra, cílená pro jednoho hráče, kdy objekty ve hře dokáže ovlivnit jen on, bude moci nabídnout stažení všech herních skutečností předem a umožnit hraní *offline*.

Aplikace umožní hráčům zobrazovat objekty a úkoly a manipulovat s nimi dle jejich nastavení a své vlastní polohy.

Návrh komunikace se systémem by měl být zpracován tak, aby se do aplikace posílala jen ta data, která jsou klíčová pro další proces. Nesmí se tedy stát, že se při připojení začnou načítat všechny objekty, informace a úkoly ve hře (pokud uživatel nenastaví jinak). Nejprve by se měly načíst jen ty úkoly, které jsou v blízkosti hráče, a to jen ty nejdůležitější informace o nich. Až po interakci s objektem (například kliknutí), se doplní informace o detailech a úkolech, které se k objektu pojí.

3.1.4 Uživatelské rozhraní - mobilní aplikace

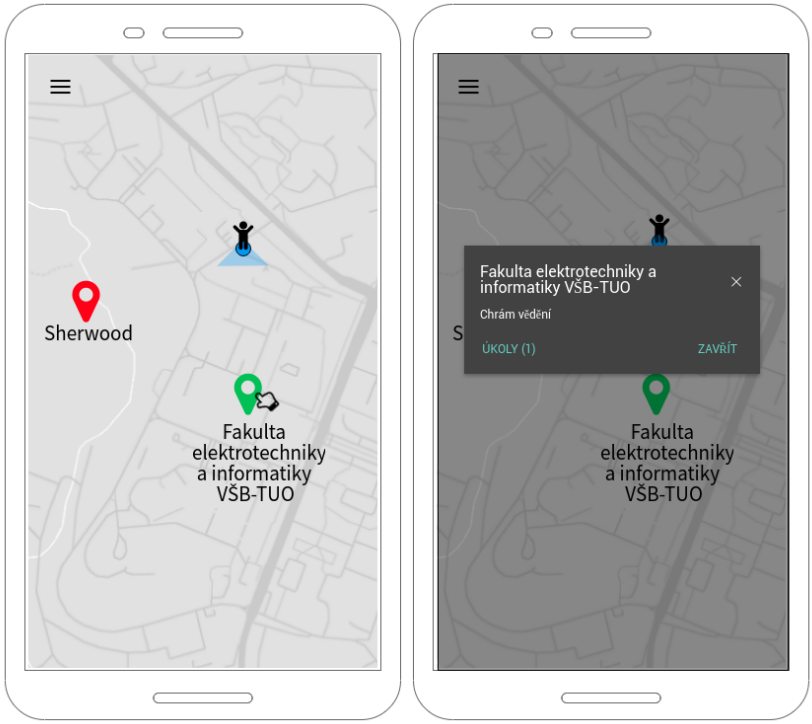
Při návrhu uživatelského rozhraní jsem postupovala podobně, jako při návrhu webového editoru, mapa s objekty by tedy měla být v centru dění. Vzhledem k omezení velikosti displeje je třeba odstranit z pohledu všechny informace, které nejsou k hraní relevantní. Zobrazeny by měly být hlavně objekty s místem, kde se nacházejí, popřípadě základní informace o nich, například název. Více informací se uživatel dozví po kliknutí na daný objekt, čímž se otevře dialogové okno s detaily.

Návrh, jak by mohla aplikace vypadat, si lze prohlédnout na obrázku 2 na straně 22.

3.2 Možnosti hry

Možnosti hry jsou dány specifikací herních vlastností ve webovém editoru. Uživatel zde bude mít možnost vytvořit, upravit nebo smazat hru. Při vytváření nebo editaci hry bude moci nastavit vlastnosti, jako je název hry, popis, rozsah zobrazovaných objektů a podobně. Rozsah zobrazovaných objektů bude sloužit pro uživatele mobilní aplikace. Tímto nastavením se omezí maximální plocha, která se může hráči na daném místě načíst. Pokud se rozsah nespecifikuje, hráč by měl vidět všechny objekty.

Zde by měl uživatel také nastavit, jestli zamýšlí vytvoření hry pro jednoho nebo více hráčů. To by se mělo stát stěžejní informací pro *offline* režim, popsaný níže.



Obrázek 2: Návrh rozhraní mobilní aplikace

3.3 Možnosti herních objektů

Možnosti herních objektů jsou, podobně jako možnosti hry, specifikovány nabízenými vlastnostmi ve webovém editoru. Musí být možné specifikovat polohu objektu, na které se bude hráčům vykreslovat a musí být možné na něj navázat akci (úkol), která se k němu bude pojit. Také se zde budou určovat speciální vlastnosti každého objektu, jako je oblast kolem zadané zeměpisné šířky a délky, ze které bude objekt přístupný hráči, nebo, jestli je objekt skrytý a nejprve bude nutno jej speciální herní akcí objevit.

3.4 Úkoly

Úkoly ve hře jsou to, co ji dělá zajímavou. Představují jednotlivé akce, které je potřeba vykonat, aby hráč něco získal, ať už je to herní objekt, souřadnice herního objektu nebo zvýšení herního skóre. Úkoly se vážou k objektům.

Typy možných úkolů jsou pevně definovány, tedy omezeny, avšak úkolů určitého typu může existovat nepřeberné množství. Typy úkolů si lze představit jako vzory, které mají definované určité chování, ale data v nich mohou být různá. Jedním z příkladů takového typu úkolu může být úkol "Hádanka". Hádanka je „drobný slovesný projev v podobě hříčky, která v náznaku předkládá určitý problém, k jehož řešení lze dospět důvtipem.“[13]. Vzor pro tento úkol bude tedy muset vždy obsahovat popis, ze kterého bude hráč vyvozovat řešení, a správnou odpověď,

ke které je zamýšleno, aby se hráč dostal. Na editorovi pak už bude, jaké úkoly typu otázka - odpověď vymyslí.

3.5 Offline režim

Offline režim je jednou z možných variant způsobu hry, kterou by aplikace mohla obsahovat. Tento způsob by však byl velmi závislý na tom, jakou formu hry si uživatel bude přát vytvořit.

Při hře pro jednoho hráče je *offline* režim možný. Podobně, jako při zmíněné hře Kód Salomon, uživateli by se mohly načíst všechny objekty předem a hra by byla provozuschopná bez připojení k internetu. Při tomto režimu by muselo být zajištěno, že jsou všechny mechanismy, potřebné pro provoz aplikace, stažené nebo implementované v aplikaci. To musí platit i pro všechny typy úkolů, které by se při editování k objektům navázaly.

Při hře pro více hráčů pak *offline* režim možný není. Může být povoleno stažení předem např. vlastních ikon nebo obrázků, pokud to bude hra dovolovat, ale při více hráčích pak neexistuje způsob synchronizace, který by zajistil konzistentní hru pro všechny.

3.6 Správa uživatelů

Uživatel se musí pro interakci se systémem a hrami registrovat. Registrace probíhá na základě zadání uživatelského jména, emailu a hesla. Registrovaný uživatel získává roli ROLE_USER.

Každá hra má pro svůj přístup tři hlavní role - OWNER, EDITOR a PLAYER. Detaily jednotlivých rolí jsou znázorněny v tabulce 1.

Rozlišení pravomocí pro různé hry je dáno kombinací ID hry a role. Při založení nové hry tak přibudou v systému tři nové role. Kupříkladu, pro hru s ID 1 by role vlastníka (OWNER) vypadala OWNER_1, pro hru s ID 11 role hráče (PLAYER) PLAYER_11.

Tabulka 1: Role uživatelů

Role	Popis	Povolené akce
-	Nepřihlášený uživatel	• registrace • přihlášení
ROLE_USER	Registrovaný uživatel	• zobrazení všech dostupných her • připojení ke hře • zobrazení her, ke kterým se uživatel připojil • vytvoření nové hry
ROLE_OWNER_id	Tvůrce hry s identifikátorem id	• smazání hry s identifikátorem id • jmenování editorů • změny vlastnických práv • editace objektů ve hře
ROLE_EDITOR_id	Editor hry s identifikátorem id	• editace objektů ve hře
ROLE_PLAYER_id	Uživatel, který se připojil ke hře s identifikátorem id	• interakce s objekty ve hře

4 Architektura geolokačního rámce

Při návrhu architektury systému jsem se zaměřila hlavně na to, aby části, zajišťující různorodost typů úkolů ve hře, byly snadno rozšiřitelné. To může zajistit rozdělení systému do jednotlivých modulů, které by na sobě byly co nejméně závislé. Takové rozdělení musí být jak na straně serveru, tak na straně uživatele ve webovém editoru i mobilní aplikaci. Ideální situace je taková, že nově naprogramovaná komponenta zajišťující průběh úkolu bude jednoduše připojitelná k systému, bez nutnosti úprav stávajícího kódu, a uživatelská rozhraní rozšířena o nové formuláře a akce. Kódy pro zpracování a vyobrazení různých typů úkolů se nesmí prolínat, každý typ musí mít své vlastní prostředí, nezávislé na prostředí typů jiných.

Také je třeba myslet na hru v reálném čase a snahu o to, mít stále ta nejaktuálnější data. Pro urychlení komunikace mezi mobilní aplikací a zbytkem systému jsou nezávislé moduly ideální. Každý z nich může být navázán na vlastní datový zdroj a mít vlastní rozhraní pro komunikaci, což zajistí, že se v mnoha případech může mobilní aplikace dotazovat přímo konkrétního modulu, bez nutnosti "probublávání"velkým systémem a blokováním dotazů na jiné části stejného systému.

S přihlédnutím k trendům posledních let je také zapotřebí vytvořit takovou architekturu, která bude moci být nasazena na jednu z cloudových služeb. To výrazně pomůže k jednoduché škálovatelnosti softwaru, a díky mnoha technologiím, které dnes cloudové služby nabízejí, může být zátěž na systém jednoduše regulována.

4.1 Architektura orientovaná na služby

"Things should be made as simple as possible, but no simpler."– Albert Einstein

Jak již Albert Einstein před mnoha lety (pravděpodobně[35]) řekl, věci by se měly dělat tak jednoduché, jak jen to je možné, jednodušší už ale ne.

Zmíněné požadavky na systém naplňuje hojně využívaný princip, který je označován jako Architektura orientovaná na služby (*Service Oriented Architecture, SOA*). Modulárnost zde zajišťují komponenty nazývané služby. Tyto služby mohou být vytvořeny za pomocí mnoha různých programovacích jazyků a technologií, ale stále spolu mohou komunikovat. Každá z nich totiž navenek vytváří na technologii nezávislé rozhraní, které ostatní služby využívají, a mohou si tak mezi sebou posílat zprávy[12].

Každá služba zastává určitou činnost. Díky tomu, že je působnost každé z nich přesně vymezena rozhraním, které nabízí, snadno se dá vyhnout stvoření velkého komplikovaného systému a potřebná funkcionalita je šikovně zapouzdřena ve službách, které jsou ve svém běhu zcela samostatné.

4.2 Architektura mikroslužeb

V souvislosti s architekturou, která je rozčleněna do více malých nezávislých komponent, se lze také setkat s pojmem *Microservice Architecture*, do češtiny přeloženo jako *Architektura mikroslužeb*. Tato architektura nemá formální definici, avšak v prostředí vývoje softwaru je to čím dál více rozšířenější pojem. Vzhledem k tomu, že není přesně definována, neexistuje přesné vymezení rozdílů mezi architekturou mikroslužeb a architekturou orientovanou na služby. Mikroslužby staví na stejných principech, jako SOA.

"The problem, however, is that SOA means too many different things."–Martin Fowler[9]

Problém je však v tom, že SOA znamená až moc různých věcí. Dle výroků Martina Fowlera si lze architekturu orientovanou na služby představit v mnoha kontextech. Jedním z nich je pojetí SOA jako vystavení softwaru pomocí webových služeb. Jiný klade důraz na rozdělení služeb, zajišťujících logiku, a dat, která spojí dohromady až prezentační vrstva. Další kontext je založen na principu komunikace, který podporuje asynchronní zpracování dat[10].

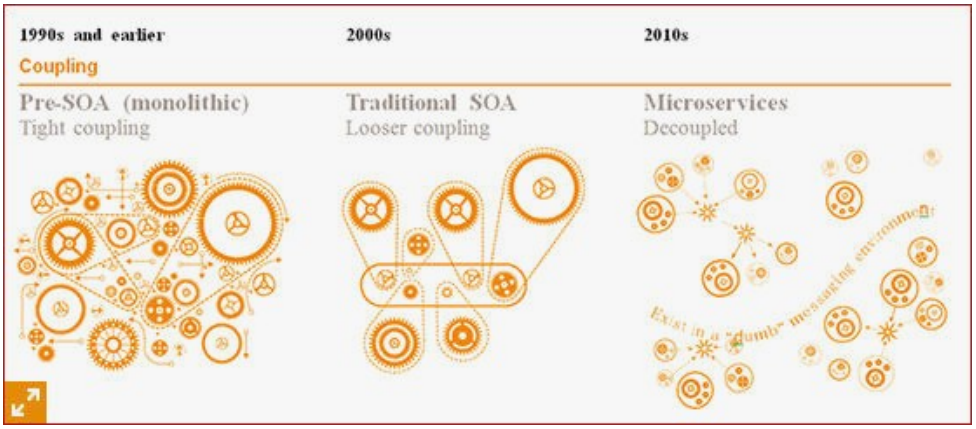
Patrným rozdílem mezi těmito dvěma principy se zdá být řešení komunikace mezi službami. S architekturou SOA se pojí pojem Podniková sběrnice služeb (*Enterprise service bus, ESB*). Ta je orchestrátorem spojení mezi službami, řídí tok dat a transformuje data tak, aby byla přístupná všem službám s rozdílnými rozhraními. Služby jsou rozděleny tak, aby zajišťovaly oddělení logických bloků, ze kterých se systém skládá. Komunikační mechanismus tak musí být nějakým způsobem "chytrý", aby dokázal využívat více služeb k vyřešení jednoho požadavku.

Přístup architektury mikroslužeb se snaží oprostit od závislosti na logice komunikačního kanálu a raději využívá technologie, které zprávy pouze přebírají a směrují na určité místo. Příkladem takové komunikace může být technologie *RabbitMQ*, která pouze zajišťuje asynchronní doručování zpráv mezi koncovými body. O logiku se starají právě koncové body každé z mikroslužeb. Každá služba tak plní nějakou funkčnost a je schopna odbavovat požadavky sama za sebe.

Ukázka, jak si lze rozdíly architektur představit, je na obrázku 3 na straně 27. První v pořadí je zde znázorněna monolitická architektura, kde je všechna funkcionalita na jednom místě a silně provázána. Dále je vyobrazena typická architektura zaměřená na služby, které jsou spjaty s komunikačním kanálem. Napravo pak můžeme vidět představu architektury mikroslužeb, které jsou od sebe i komunikačního kanálu oddělené.

Ačkoli lze z mnoha zdrojů vyčíst určité rozdíly mezi oběma architekturami, obě se snaží rozdělit systém do nezávislých komponent, aby zabránily vzniku velkým monolitickým aplikacím. Terminologicky je tak možné označit architekturu mikroslužeb jako konkrétnější formu architektury orientované na služby.

Ať již použijeme označení služeb či mikroslužeb, výhodou takto distribuovaného systému je v případě potřeby přidání, úprava nebo výměna služeb bez nutnosti zásahu do stávajícího systému. Je možné manipulovat s jednou částí bez toho, aniž by to ohrozilo jiné služby, které tak není třeba znova sestavovat, někdy ani zastavovat.



Obrázek 3: Porovnání monolitu, SOA a mikroslužeb
[29]

Hlavní nevýhoda spočívá v tom, že je zapotřebí využít další technologii, která musí zajišťovat komunikaci mezi všemi částmi systému. Při závažné chybě v systému, kdy komunikační kanál ukončí svou činnost, je bohužel celý systém paralyzován.

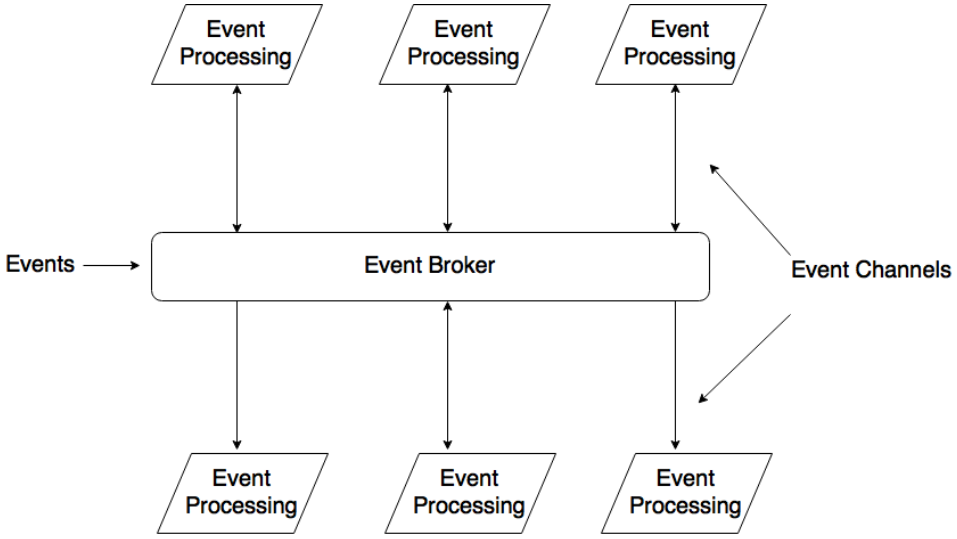
4.3 Komunikace

Na komunikaci mezi jednotlivými komponentami systému závisí mnoho. Špatně navržená komunikace se může výrazně projevit na rychlosti a výkonu systému, přelévání velkého množství dat dokáže zahltit systém a zpomalit nebo zablokovat přenášení zpráv. Efektivní komunikace proto musí být založena na zasílání asynchronních zpráv, aby nedocházelo ke zbytečnému blokování systému při čekání komponent na odpovědi.

4.4 Zprostředkovatel

Použitý princip, korespondující s distribuovaným prostředím nezávisle spolupracujících komponent, zakládá na návrhovém vzoru *Broker* (*Zprostředkovatel, Jednatel*)[11]. Tento vzor si dává za cíl právě řešení komunikace mezi komponentami tak, aby jedna komponenta při zaslání zprávy nemusela znát cestu ke komponentě druhé. Izoluje tak jednotlivé části systému a zastává funkci směrovače zpráv. Zároveň se takto také smazává tradiční rozdělení systému na části klientské a části serverové, kdy jedna komponenta může za běhu působit zároveň jako server i jako klient.

Takto koncipovaný systém vyžaduje přidání komponent, které zajišťují komunikaci mezi aplikacemi a zprostředkovatelem. Zprávy od aplikací se nejdříve dostanou k prostředníkovi (*proxy*), který formu zprávy "přeloží" do formátu, který je přenositelný zprostředkovatelem. Aplikace, kterým je zpráva určena, pak musí za pomocí svého prostředníka zprávu nejdříve transformovat tak, aby odpovídala jejich vnitřní struktuře, než s ní budou moci pracovat.



Obrázek 4: Událostmi řízená architektura a zprostředkovatel

4.5 Událostmi řízená architektura

Aby bylo možné využít architektury nezávislých služeb i jednoduchého komunikačního kanálu, je nutné přenášené zprávy koncipovat podle nějakých pravidel tak, aby každá služba byla schopna formát dat přeložit a s daty dále pracovat. Zároveň také musí být schopna vytvořit zprávu bez toho, aniž by znala přesný cíl doručení. Z tohoto důvodu jsem se rozhodla využít techniku, která řídí tok dat pomocí událostí.

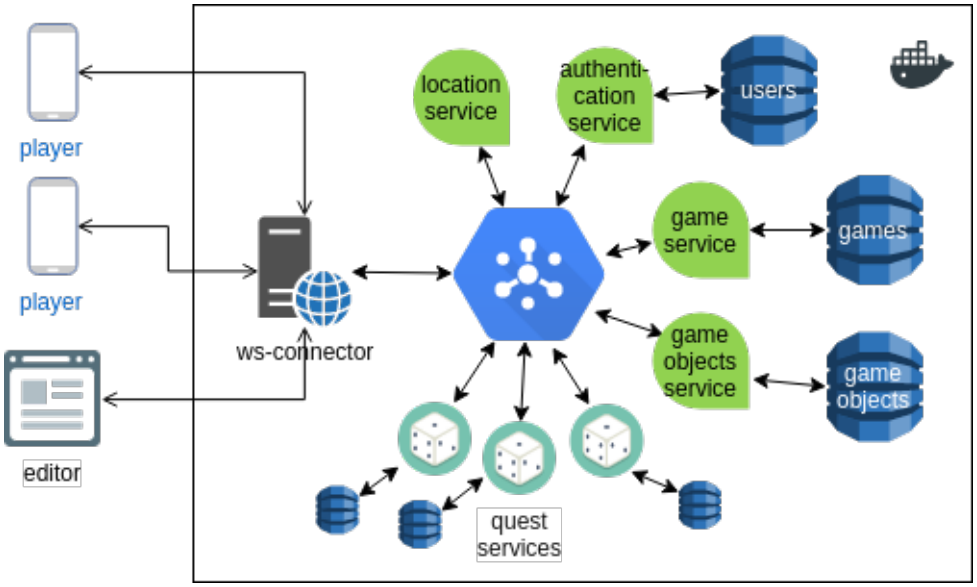
Jak Martin Fowler opět podotýká, když lidé mluví o událostech, v mnoha případech myslí mnoho rozdílných věcí[34].

V podání této diplomové práce bude událost považována za zprávu, kterou budou přijímat a zasílat jednotlivé služby, popř. mobilní aplikace a webový editor. Každá takováto událost bude složena ze dvou hlavních částí:

1. *Typ události*: Tato informace bude stěžejní při každé příchozí zprávě. Každá komponenta musí implementovat mechanismus s přístupem ke všem možným událostem, které je daná komponenta schopna zpracovat, a využít tento mechanismus k vyvolání příslušné akce.

2. *Data*: Druhou částí události jsou data. Pomocí typu události se rozhodne, jaké akce se budou provádět. Tyto akce budou předpokládat určitý formát dat a jejich součástí bude získání potřebných informací z nich.

Zamýšlený vztah architektury komponent, událostí a zprostředkovatele znázorňuje obrázek 4 na straně 28. [5]



Obrázek 5: Celková architektura systému

4.6 Jednotlivé komponenty

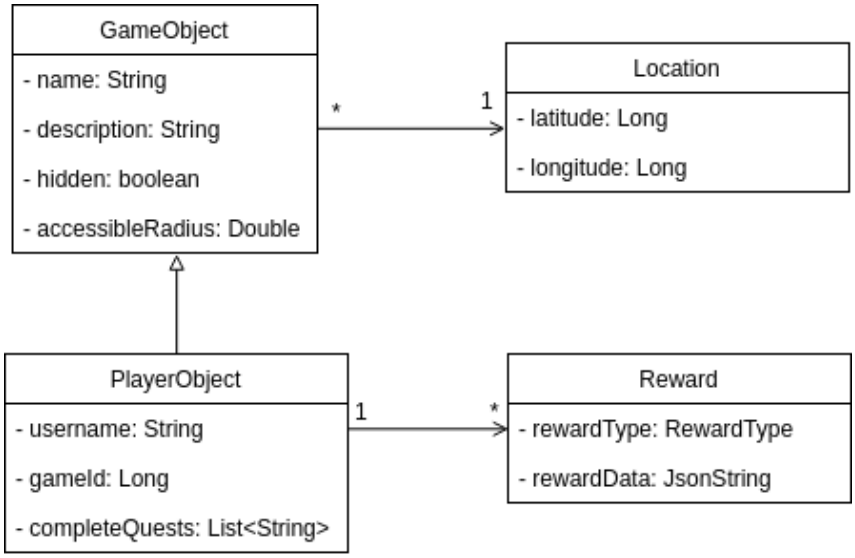
Prvním krokem k vytvoření celkové architektury systému bylo vymyslet možná rozčlenění jednotlivých komponent. Služby musejí být rozděleny tak, aby, pokud možno, byla potřeba komunikace s ostatními udržována na minimu. Výsledné rozdělení znázorňuje obrázek 5 na straně 29. Jádrem architektury je objekt, ke kterému jsou všechny ostatní objekty připojeny. Ten znázorňuje zprostředkovatele pro předávání zpráv mezi službami. Význam jednotlivých částí a služeb bude popsán níže.

Práce s mnoha komponentami může být náročná na údržbu, proto musí být ve výsledku použita určitá technologie, která je schopna spravovat všechny komponenty a zajistit jejich provoz.

4.6.1 Game service

Část systému, nazvaná *Game service*, je určena pro spravování veškerého nastavení pro konkrétní typ hry. Jedná se o "statické" informace, které určují hlavně strukturu jednotlivých her. Zde se pracuje s nastaveními, která si uživatel při vytvoření hry zvolil, jako je maximální vzdálenost uživatele od objektu, aby jej byl schopen vidět na mapě.

Zprvu se oddělení služby *Game service* a *Game objects service* zdálo nevýhodné a vypadalo to, že provázání těchto dvou služeb bude vysoké. Avšak tím, že herní akce jsou zajišťovány jednotlivými komponentami typů úkolů, odstínila se herní logika od objektu hry (ačkoli to může znít zvláštně). *Game service* zde figuruje hlavně jako správce specifikací jednotlivých her. Objekty konkrétních her tak stanovují hranice a povolené akce, které je možné v daném typu hry využít.



Obrázek 6: Herní objekt a objekt hráče

4.6.2 Game objects service

Tato služba spravuje vše, co se týče herních objektů. Samotná hra, popsaná v předchozí kapitole, nemá o jakýchkoli konkrétních objektech, které jí náleží, žádné ponětí. Proto sám objekt musí vědět, pro kterou hru byl vytvořen a kam patří. To však nemusí být definitivní, teoreticky každý objekt může být použit v jakékoli jiné hře.

Herní objekty mohou mít mnoho vlastností. Stěžejní z nich je bezpochyby zeměpisná šířka a výška, ve které se nachází. Podle těchto informací se objekty filtrují a vykreslují. Každý objekt nese také informaci o tom, z jaké vzdálenosti je možné s ním manipulovat. Ne každý objekt ale musí mít určenou polohu. Takovéto objekty se nacházejí v inventářích ostatních objektů, mohou například zaručovat postup ve hře, nebo reprezentují jakousi odměnu.

V rámci toho, že herních objektů a jejich modifikací může existovat snad nekonečně mnoho, rozhodla jsem se pro generické řešení. Místo vytváření mnoha různých tříd s jinými vlastnostmi je použita jen jedna třída, která obsahuje vlastnosti všechny. Při vytvoření nového objektu nebo načtení objektu stávajícího určují data, jaký charakter objekt má a jak se bude chovat. Pokud nebude nastavena vlastnost, kupříkladu možnost skrytí objektu za určitých podmínek, automaticky se předpokládá, že objekt tuto vlastnost nemá.

Speciální objekt ve hře je reprezentován třídou *PlayerObject*. Tento objekt je odvozen od základní třídy *GameObject* a reprezentuje postavu uživatele ve hře. Diagram jejich vztahu je zobrazen na obrázku 6 na straně 30.

Stav hry, ve kterém se uživatel zrovna nachází, je vázán ne přímo na samotného uživatele, ale právě na tento herní objekt *PlayerObject*. Dalo by se říci, že vnitřně tedy uživatel nehraje sám za sebe, ale za postavu, která je na něj vázána. Takováto postava pak může jednoduše využívat všechny herní vlastnosti, které se vztahují i na obyčejné objekty, jako je inventář nebo poloha.

Navíc si však uchovává všechny akce, které se kdy s postavou staly, lze tak přes ní získat třeba seznam všech úkolů, které uživatel již vyřešil.

Díky tomu, že se stav hry uživatele uchovává přímo v jednom z herních objektů, otevírá se možnost vytvoření specializované postavy. Některý typ her nemusí vyžadovat žádné speciální vlastnosti a po uživateli nepožaduje kroky pro vytvoření postavy. Při tomto typu hry je postava vytvořena automaticky a slouží hlavně pro ukládání stavu hry. Jiné typy hry ale mohou požadovat, aby si uživatel zvolil některé ze speciálních vlastností, které budou ovlivňovat jeho výkon ve hře.

Příkladem může být herní objekt zamčených dveří. Řekněme, že máme možnost specializace zloděje a válečníka. Uživatel, který si jako svou specializaci zvolil zloděje, dokáže otevřít dveře speciálním paklíčem. Válečníkovi bude paklíč zbytečný, avšak dokáže dveře otevřít silou. Za dveřmi se, podle toho, jakou má hráč specializaci, mohou skrývat odlišné věci. Válečník může získat novou sekeru, zloděj vzácné cennosti.

4.6.3 Authentication service

Autentizační služba se stará čistě o autentizaci uživatelů a ostatních služeb. Vyřizuje požadavky o registraci a přihlášení a ukládá uživatelské přístupy.

4.6.4 Location service

Lokalizační služba se stará o vyřizování požadavků souvisejících z polohou uživatele.

Při připojení nového uživatele zařizuje, aby se k uživateli dostaly objekty v takové vzdálenosti, jaká je buď uživatelem, nebo konkrétní hrou specifikována. Uživatel periodicky zasílá této službě informace o své poloze. Lokalizační služba pak pošle dotaz službě obstarávající herní objekty a zjišťuje, zda-li se v rozsahu uživatelova "dohledu" nacházejí nové objekty.

Stará se také o to, aby uživateli nebylo zasíláno více informací, než je nezbytně nutné. Sama si identifikátor objektů, které již uživatel obdržel, ukládá do své paměti a při požadavcích nejprve filtruje, jestli se neobjeví objekty nové, které je nutné klientovi zaslat.

4.6.5 Quest services

Další částí systému jsou služby, které se starají o vytváření, úpravu, vykonávání a mazání úkolů. Aby byla zajištěna co největší modularita této části, každý typ úkolu a činností s ním souvisejících je zabalen do vlastní služby. Co všechno musí tyto služby poskytovat a splňovat bude shrnuto později, v implementační části.

4.6.6 Web editor, player

Poslední částí, vyobrazenou na diagramu, je webový editor a objekty *player*, které znázorňují hráče a jejich mobilní aplikaci. Pro tyto komponenty musí existovat speciální *connector*, který

je dokáže připojit ke zprostředkovateli, čímž odstíní samotné uživatelské aplikace od nutnosti implementace konkrétních postupů ke komunikaci se zbytkem systému.

5 Implementační část

Tato část diplomové práce se bude zabývat konkrétními technologiemi, projekty a postupy, které byly pro vývoj geolokačního rámce použity. Použité technologie budou popsány, uvedeny důvody aplikace daných postupů a detailněji rozebrány některé části systému a použití specifických technik.

V části o mobilní aplikaci, webovém editoru a jednotlivých službách nalezneme podkapitoly *rozšiřitelnost*. V těchto kapitolách bude popsán způsob, jakým je řešena modulárnost jednotlivých typů úkolů a co je zapotřebí pro rozšíření těchto částí o další typy.

Konkrétní popis všech funkcionalit, které jsou v rámci implementovány, je popsán v příloze *B*.

5.1 Mapy

Nepostradatelným pomocníkem geolokačních her jsou určitě mapy. V dnešní době máme na výběr ze dvou hlavních projektů, které velmi obsáhlé mapové podklady nabízejí - *Google Maps* a *OpenStreetMap*.

Jako hlavní rozdíl mezi těmito dvěma poskytovateli bych označila "myšlenkový" přístup. *Google Maps* jsou komerčním projektem vlastněným společností *Google* a všechna data a techniky, které můžeme jako vývojáři pracující s *Google Maps API* využít, jsou jen ty, která nám *Google* zpřístupní. *OpenStreetMap*, na druhou stranu, pracuje s koncepcí otevřeného softwaru, kdy jsou data poskytována pod licencí *Open Database Licence*. To nám dává svobodu veškerá řešení *OpenStreetMap* převzít a volně je upravit k obrazu svému. V obecném porovnání je *Google Maps API* oproti *OpenStreetMap* jednodušší na použití, ale méně flexibilní[22].

V implementaci geolokačního rámce jsem využila *OpenStreetMap*, právě z důvodu otevřených dat. *Google Maps API* poskytuje určitou funkcionalitu široké veřejnosti, bez nutnosti zakoupení vyšších kvót, avšak po naplnění stanoveného limitu dotazů na jejich servery se tato funkcionalita zablokuje [21]. Toto chování by mohlo vyvolat problémy při testování geolokačního rámce.

5.1.1 OpenStreetMap

OpenStreetMap je komunitním projektem, který si klade za cíl vytváření a zpřístupňování geografických dat z celého světa a pro celý svět. Jeho princip lze přirovnat k principu fenoménu *Wikipedia*, kdy obsah přidávají a udržují dobrovolníci. Členem komunity se může stát kdokoli, každý tak k projektu může přispívat novými daty či aktualizacemi.

Pro práci s tímto projektem jsem využila dvou knihoven, které poskytují rozhraní k přístupu a úpravě map. Je to knihovna *osmdroid*, využita v souvislosti s mobilní aplikací, a *leaflet*, použita pro vykreslování a editaci map ve webovém editoru. Ve výsledném řešení jsou zvolené mapy implementovány se snahou o to, aby byly, v případě potřeby, snadno zaměnitelné za jiný mapový zdroj. Zvolené knihovny komunikují se zbytkem kódu prostřednictvím rozhraní, které zajišťuje jejich izolaci.

5.2 Mobilní aplikace

5.2.1 Android

Mobilní aplikace byla vyvíjena pro systém *Android 7.0* za použití programovacího jazyka *Java*. K jejímu testování bylo použito mobilní zařízení *Huawei VNS-L21*.

Platforma *Android* byla zvolena z důvodu jejího masivního využití. Dle společnosti *Gartner*, americké firmy zabývající se výzkumem a dohledem v oblasti informačních technologií, bylo zjištěno, že v první čtvrtině roku 2017 přibližně 86% prodaných chytrých mobilních zařízení využívalo právě operační systém *Android*.

5.2.2 Uživatelské rozhraní

Uživatelské rozhraní je složeno z několika *Aktivit*. *Activity* je třída používající se pro implementaci jedné logické části aplikace, která zajišťuje jedinou činnost. V aplikaci je implementováno několik *aktivit*:

1. *MainActivity*: zajišťuje zobrazení úvodní obrazovky aplikace a inicializaci spojení se systémem
2. *LoginActivity*: *aktivita* zajišťující zobrazení formuláře, který umožňuje přihlášení, i provedení samotného přihlášení
3. *WelcomeActivity*: tato *aktivita* zajišťuje zobrazení formuláře pro výběr hry, které se chce uživatel zúčastnit
4. *GameAreaActivity*: *aktivita* zajišťující vykreslení hracího pole, nachází se zde hlavní rozhraní pro manipulaci s objekty

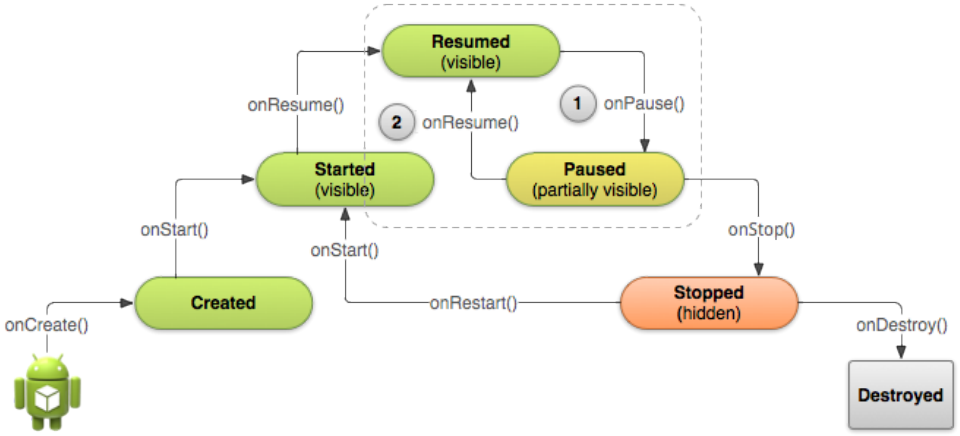
Každá *aktivita* má svůj vlastní životní cyklus. Ten je reprezentován metodami, které je možné přepsat a přizpůsobit vlastním potřebám. Je to například metoda *onCreate()*, která se provádí při vytváření *aktivity*, *onStart()* prováděna při startu *aktivity*, *onPause()* řešící stav, kdy *aktivita* stále běží, ale je kupříkladu překryta jinou *aktivitou*, a další.[33] Všechny části životního cyklu jsou znázorněny na obrázku 7 na straně 35.

Součástí *aktivity GameAreaActivity* je *MapFragment*. *Fragment* je část uživatelského rozhraní, kterou je možné vložit do jakékoli *aktivity*. S *aktivitou* je pevně svázána a nedá se využít mimo ni. Má také vlastní životní cyklus, ale ten je závislý na cyklu *aktivity*, ve které se nachází.

Třída *MapFragment* je rozhraním pro použití knihovny *osmdroid*, která je použita jako zdroj mapových podkladů, interakce s objekty a geolokace.

5.2.3 osmdroid

osmdroid je knihovna, která nahrazuje, možná známější, *MapView* projekt společnosti *Google*, v kontextu *OpenStreetMap*. Obsahuje modulární systém pro zobrazování různých druhů takzva-



Obrázek 7: Životní cyklus *aktivity*

ných *mapových dlaždic*, což jsou bitmapové čtverce uspořádané v mřížce pro zobrazení mapy[23], umožňující zobrazovat různé verze map v závislosti na jejich významu, ať už mapová data cyklotras nebo tras námořních. Podporuje také vytváření dodatečných překryvných vrstev, které slouží k přidávání ikon, vyznačování cest nebo zobrazování tvarů[20].

Základem pro vykreslení mapy je vytvoření objektu třídy *MapView*, který knihovna poskytuje, a jeho předání pro zpracování při vytváření *view* pro *fragment*.

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
    savedInstanceState) {
   mMapView = new MapView(inflater.getContext());
   return mMapView;
}
```

Výpis 1: Inicializace *MapView*

V metodě *onActivityCreated()* pak můžeme specifikovat jednotlivé detaily mapy, které chceme nebo nechceme v aplikaci vykreslit.

```
@Override
public void onActivityCreated(Bundle savedInstanceState) {
   super.onActivityCreated(savedInstanceState);
   mMapView.setBuiltInZoomControls(true);
   mMapView.setTileSource(TileSourceFactory.PUBLIC_TRANSPORT);

   this.mCompassOverlay = new CompassOverlay(context, new
       InternalCompassOrientationProvider(context), mMapView);
   this.mCompassOverlay.enableCompass();
   mMapView.getOverlays().add(this.mCompassOverlay);
```

```
}
```

Výpis 2: Detaily vykreslení mapy

V ukázce výše kód specifikuje, že v zobrazení mapy se mají objevit tlačítka pro přiblížení a oddálení mapy, vzhled mapy má být postaven na základě veřejné dopravy a součástí mapy má být také kompas. *mMapView.getOverlays().add(...)* nám umožňuje vkládat do mapy nové překryvné vrstvy, v tomto případě právě kompas.

Pro stávající implementaci je však důležitější vrstva *ItemizedIconOverlay*. Využívá objektů třídy *OverlayItem*, které do ní můžeme vložit a zobrazit. Takto se v aplikaci vykreslují body, se kterými pak může uživatel interagovat. Následující kód ukazuje, jak se přidání jednotlivých bodů může provést.

```
public void addPoint(GameObject object) {
   GeoPoint point = new GeoPoint(object.getLocation().getLattitude(), object.
      getLocation().getLongitude());
   OverlayItem item = new OverlayItem(object.getId().toString(), object.
      getDescription(), point);
   ((ItemizedIconOverlay) this.mItemizedOverlay).addItem(item);
}
```

Výpis 3: Přidání bodu do mapy

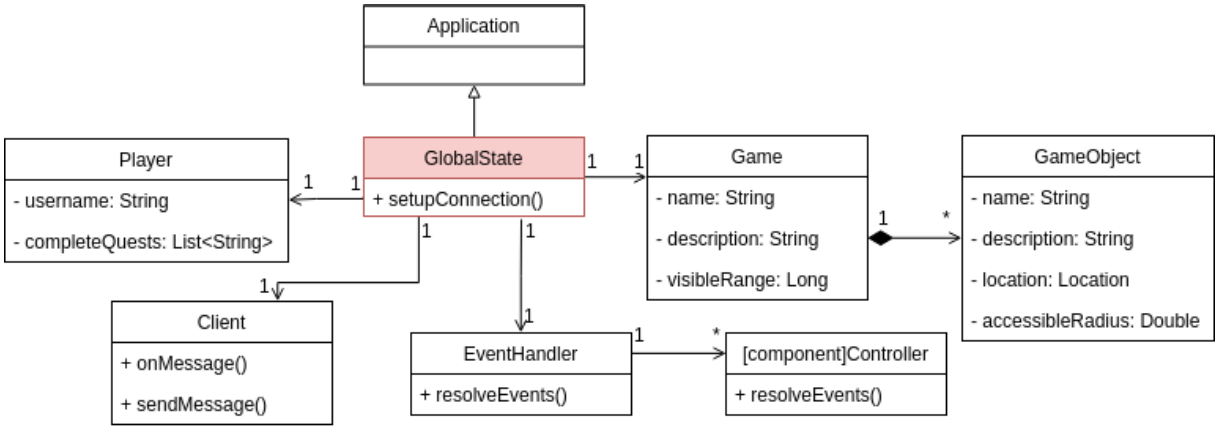
5.2.4 Rozložení tříd

Mobilní aplikace je navržena tak, aby v případě změny UI nebo přidání nových typů akcí (úkolů) poskytovala rozhraní, se kterým mohou nové komponenty komunikovat.

Jádrem aplikace je třída *GlobalState*. Objekt této třídy poskytuje všechnu důležitou funkcionalitu i logiku herních akcí. Ukládá si data o aktuální hře, objektech, které byly v průběhu hry načteny, i o stavu hry prostřednictvím objektu hráče. Zjednodušený třídní diagram popisující vztahy jednotlivých komponent znázorňuje obrázek 8 na straně 37

Díky tomu, že *GlobalState* dědí z nativní třídy systému Android *Application*, je tak přístupný všem UI komponentám, které potřebují znát aktuální data hry. Přístup k němu je k vidění na následujícím výpisu:

```
public class LoginActivity extends Activity implements UINotificationListener{
  GlobalState globalState;
  @Override
  public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
     globalState = ((GlobalState) getApplicationContext());
  }
```



Obrázek 8: Znázornění vztahů tříd s třídou *GlobalState*

```
}
```

Výpis 4: Ukázka přístupu z *LoginActivity* k objektu *GlobalState*

Názory na existenci třídy, která k uložení globálního stavu využívá třídu *Application*, se liší. V prostředí internetových komunit vývojářů lze determinovat dvě základní frakce:[32]

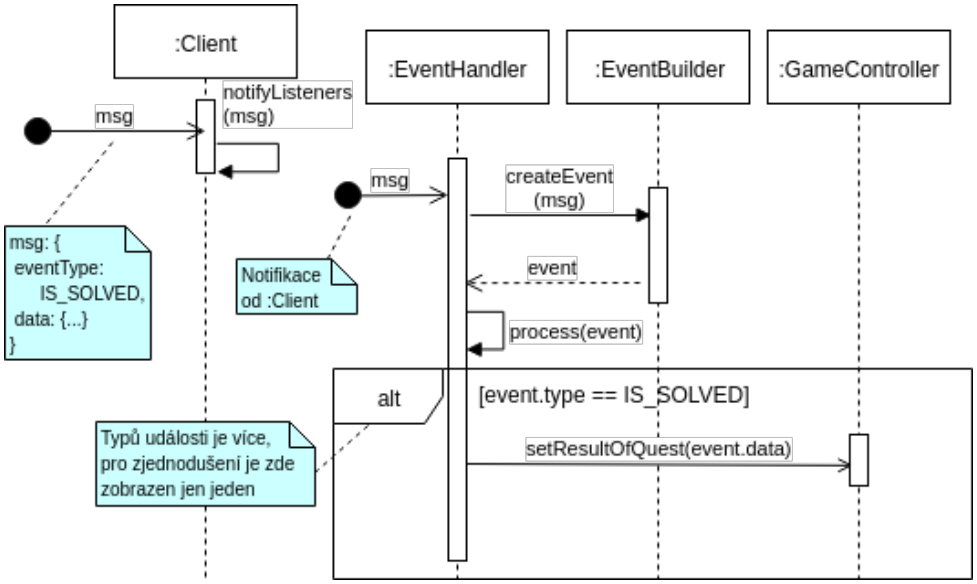
1. Application context: Využití třídy *Application* k získání globálního kontextu
2. Singletons: Vytvoření více objektů dle návrhového vzoru *Jedináček* (*Singleton*)[4], ve kterém se ukládají data, která je potřeba sdílet

Oba přístupy mají svá pro a proti. *Singleton* je často považován za *anti-pattern*, komponentu, která dělá kód více komplexnějším a snižuje celkovou testovatelnost systému [31]. Oproti využití *Application* ale dokáže jednotlivá data oddělit od sebe a každý *Singleton* tak může spravovat jinou, logicky oddělenou část, namísto uložení potřebných dat na jednom místě, jak to dělá *Application*. *Application* kontext se sám o sobě dá považovat za *Singleton*, ale je spravován samotným rámcem, má definovaný přístup k němu i svůj životní cyklus[30]. Proto jsem se vydala cestou použití třídy *Application*, využití toho či onoho principu je však na zvážení.

5.2.5 Ukázka zpracování příchozí události

Při nové příchozí zprávě se data nejprve zpracovávají a přenášejí do objektu třídy *Event*, přičemž je určen typ události. Ten poté putuje k objektu třídy *EventHandler*, kde se dle daného typu události vyvolá korespondující objekt třídy *Controller*. Každý *Controller* implementuje rozhraní *UINotificator* a upozorňuje uživatelské rozhraní na změny podle návrhového vzoru Pozorovatel (*Observer*).[4]

Kupříkladu, řeší-li uživatel hádanku (*RiddleQuest*) a zašle systému řešení k vyhodnocení, systém mu v odpovědi pošle typ události *IS_SOLVED* s informacemi o tom, jestli uživatel hádanku vyřešil nebo ne, popřípadě informace o odměně. Činnost znázorňuje sekvenční diagram



Obrázek 9: Přijetí nové události v Android aplikaci

na obrázku 9 na straně 38. *EventHandler* při této události využije objekt třídy *GameController* a vyvolá metodu *setResultOfQuest()*, které předá patřičná data.

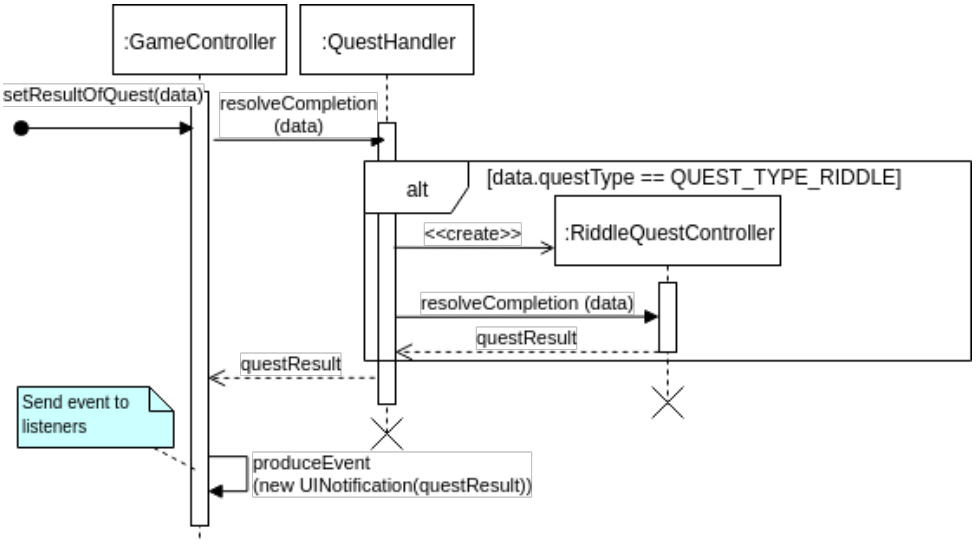
Zde se využije třídy *QuestHandler*, která, dle typu úkolu, vytvoří nový objekt odpovědi (diagram 10 na straně 39). Pracujeme-li s úkolem typu *RiddleQuest*, vrátí se objekt *RiddleResult*. Tento objekt implementuje rozhraní *QuestResult*, se kterým *GameController* umí pracovat a odděluje jej tak od konkrétního typu úkolu.

Po získání objektu *QuestResult* je v objektu *GameController* vytvořena nová událost, tentokrát notifikace pro uživatelské rozhraní skrze třídu *UINotification* s kladnou nebo zápornou odpovědí na akci uživatele, a všichni posluchači tohoto objektu jsou notifikováni. Zde je to konkrétně objekt třídy *MapFragment*, který se stará o vykreslování všech skutečností souvisejících s mapou.

Notifikace je zpracována podobně, jako předchozí událost *IS_SOLVED* v objektu *EventHandler*, a je vybrána správná metoda, kterou je nutno provést. Protože se jedná o akci související s úkoly, je do akce povolán objekt *QuestManager*. Tento objekt už opět musí pracovat s konkrétními objekty konkrétních typů úkolů, aby bylo zajištěno, že uživateli zobrazí správný dialog. Při tomto typu úkolu (*RiddleQuest*) je vyvolán objekt třídy *RiddleQuestUIManager*. Ten zhodnotí, je-li třeba vytvořit dialog hádanky značící úspěch, který uživateli potvrdí, že jeho odpověď byla správná, nebo dialog o selhání, tedy, že se uživatel zmýlil.

5.2.6 Rozšiřitelnost

Aby bylo přidání nového typu úkolu co nejjednodušší, všechna funkcionalita pro daný typ by měla být obsažena v jednom modulu. Tento modul se přidá do balíčku *diploma.geoframe.quests* (dále jen *quests*). Balíček pro úkol hádanky, použité v předchozím příkladu, tedy vypadá následovně:



Obrázek 10: Řešení události objektem třídy *GameController*

diploma.geoframe.quests.riddleQuest

V balíčku *quests* jsou specifikovaná rozhraní, která slouží pro komunikaci tříd, specializovaných pro daný úkol, se zbytkem systému. Tyto rozhraní a konkrétní implementace pro úkol *RiddleQuest* jsou vyobrazeny na třídním diagramu 11 na straně 40.

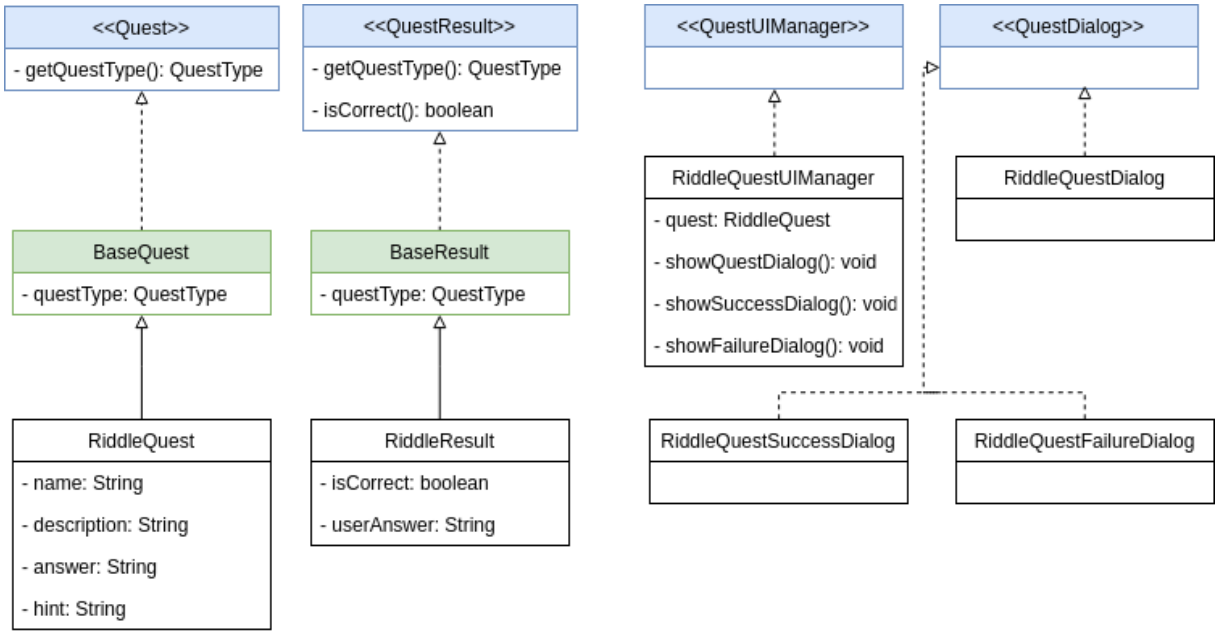
Rozhraní *Quest* musí implementovat objekt, který svými vlastnostmi reprezentuje konkrétní úkol. Rozhraní *QuestResult* musí implementovat výsledek úkolu, který značí, jestli byl daný úkol splněn nebo ne. Obě tyto třídy se musejí, v případě potřeby, umět navenek prezentovat svým unikátním typem *QuestType*. Toho je využito pokaždé, když je třeba vyvolat akci související s konkrétním typem úkolu, například zobrazení dialogových oken. Tak se děje v objektu třídy *QuestManager*. *QuestManager* musí zkontrolovat, jakého typu úkol je, a poté vyvolat patřičnou akci k zobrazení specifických uživatelských rozhraní. Zde je to konkrétně vytvoření nového objektu *RiddleQuestUIManager*.

Aktivita s objektem *RiddleQuestUIManager* komunikuje přes rozhraní *QuestUIManager*. Přes toho rozhraní je určeno, jaká činnost se má provést. Pouze pro zobrazení uživatelského rozhraní pro práci s úkolem je použita metoda *showQuestDialog()*. Zde práce aktivity končí a *RiddleQuestUIManager* se musí postarat o všechny činnosti, související se specifickým typem úkolu.

5.3 Offline režim

Jak bylo řečeno v kapitole návrhu architektury, provoz *offline* režimu může být při takto zamýšlené modulární hře problematický.

I při hře jednoho hráče můžou nastat problémy, které mohou způsobit různé typy úkolů. Při představě, že existuje typ úkolu, který potřebuje připojení k internetu, kupříkladu obsahuje URL pro načtení obrázku, nastal by problém. Možná řešení by byla:



Obrázek 11: Rozhraní, která musejí být implementována konkrétními typy úkolů

- Samotná služba pro úkol musí obsahovat mechanismus, který dokáže vyřešit závislosti na externích zdrojích, stáhnout je a dopravit k uživateli. Služba pro daný úkol by tak ale musela implementovat logiku navíc, která by nesouvisela jen s poskytováním dat a řízením stavu úkolu, ale i s přístupem k internetu a stahování závislostí.
- Existence nové *offline* služby, která by byla schopna řešit právě ty závislosti, u kterých je nutný přístup k internetu. Taková služba by odstínila služby typů úkolů od nutnosti každé zvlášť implementovat vlastní řešení pro práci s internetem. Takováto služba by ale zase musela mít povědomí o všech službách úkolů, pro které z nich je třeba řešit závislosti a jaký typ dat je nutno přenášet. To by zvýšilo obtížnost připojení nové služby pro nový typ úkolu, který by počítal s internetovým připojením na straně uživatele, protože by musela být specifikována právě v této *offline* službě.
- Zakázání daného typu úkolu pro *offline* režim.

První dvě nastíněná řešení mají své nevýhody a nejeví se jako optimální. Poslední řešení není úplně řešením, které by dokázalo ponechat daný typ úkolu v *offline* režimu hry, ale jeví se jako nejčistší a nejlogičtější. Každá služba pro typ úkolů by musela dokázat informovat, jestli je koncipována tak, aby bylo možné typ úkolu bez problému zaslat, pokud uživatel zvolí *offline* režim. Tak by se při editaci hry, s nastavením pro povolení *offline* režimu, typy úkolů závislé na internetu editorovi vůbec nenabízely, nebo by nebyly použitelné.

Reálnější řešení, které by bylo na rámec globálně aplikovatelné, není kompletní *offline* režim, ale jen část skutečností stažitelných před samotným začátkem hry. Mohou to být úkoly, které

mají jasně definovaný svůj objem dat a všechny potřebné závislosti dokáží vyřešit bez připojení k internetu, vlastní design uživatelského rozhraní nebo různé druhy mapových podkladů, bude-li v budoucnu implementace tyto prvky podporovat.

Při tomto přístupu by tak aplikace musela navíc pro jeden typ úkolu implementovat jak *online* verzi, kdy například sama zařídí načtení obrázku z internetu, tak *offline* verzi s přístupem do paměti zařízení. Do samotné aplikace by pak musela být přenesena stěžejní funkcionalita pro řešení logiky daného typu úkolu.

Zakomponování kompletního *offline* režimu do rámce ve stávající implementaci není, je však teoreticky možné, pouze ale v konkrétních hrách, které by svým nastavením a složením úkolů předcházely nutnosti využít internet.

5.4 Webový editor

5.4.1 AngularJS

Angularjs je webový MVC framework založený na programovacím jazyku *JavaScript*.

Je znám zejména pro svůj koncept "Two-way data binding", který umožňuje vkládat dynamický obsah do statického HTML souboru. V praxi to vypadá tak, že přidáním speciálních atributů k HTML značkám docílíme automatického zpracování a synchronizaci definovaných proměnných v souboru. Když se změní *view*, automaticky se změní i *model* a naopak, když se změní *model*, *view* je ihned obnoveno[24].

5.4.2 Leaflet

Leaflet je další knihovnou z řad otevřeného softwaru. Je implementována v programovacím jazyce *JavaScript* a slouží k poskytování mapových podkladů za účelem vyobrazení na webových stránkách.

Tato knihovna je ve stávající implementaci geolokačního rámce připojena přes již existující direktivu, vytvořenou za pomocí *Angularjs*, nesoucí název *angular-leaflet-directive*[25]. Díky ní je integrace map *Leaflet* neuvěřitelně snadná. Ukázku ze stávajícího kódu z HTML souboru *mapEditor.html* můžeme vidět v následujícím výpisu:

```
<main class="fullscreen map">
  <leaflet width="100%" height="100%" defaults="defaults" controls="controls"
    lf-center="ostrava" markers="markers" >
  </leaflet>
</main>
```

Výpis 5: Ukázka integrace *Leaflet* za pomocí *angular-leaflet-directive*

Zde jsou již nastaveny atributy, jako je střed mapy, odkud se bude mapa vykreslovat, nebo body zaznačené v mapě. Tyto atributy jsou volitelné. Úplné minimum pro vykreslení mapy je takovéto:

```
  <leaflet></leaflet>
```

Výpis 6: Minimální integrace *Leaflet* za pomocí *angular-leaflet-directive*

Přidáním této direktivy do kódu (samozřejmě s vyřešenými závislostmi na frameworku *Angularjs*, knihovně *Leaflet* i direktivě *angular-leaflet-directive*), se automaticky vykreslí okno s mapou. Nastavení konkrétních skutečností, jako je centrování načítané mapy, se provádí v souborech jazyka *javascript*. Stačí rozšířit konkrétní *$scope* o nový, definovaný bod, v našem případě pojmenovaný *ostrava* a přidat k direktivě atribut *lf-center=ostrava*, možný vidět v předchozím výpisu.

```
angular.extend($scope, {
   ostrava: {
      lat: 49.835556,
      lng: 18.292778
});
```

Výpis 7: JS kód pro definici bodu s *angular-leaflet-directive*

5.4.3 Rozšiřitelnost

Při přidání nového typu úkolu je třeba přidat i nové formuláře do webového editoru, skrze které je možné objekty daného typu vytvářet. Ty mají v projektu své specifické místo, konkrétně pod složkou *quests*. Zde je možné vytvořit nový modul, značící typ úkolu, a do něj vytvořit veškeré komponenty, které jsou podstatné ke specifikování a zpracování nového úkolu.

Konkrétní typ úkolu pak uživatel vybírá přes okno *questModal*, odkud je editor dále nasměrován na právě námi specifikované rozhraní, související s novým typem úkolu.

5.5 Jednotlivé služby

Všechny jednotlivé služby, vyjmenované v kapitole 4.6, jsou víceméně postaveny na stejných principech a technologiích. Pro implementaci je použit programovací jazyk *Java* s využitím aplikačního rámce *Spring*, nastavení a spuštění projektů zajišťuje *Spring Boot* a závislosti a sestavení jsou řešeny pomocí technologie *Maven*. Přístup k databázi probíhá s využitím *JPA*.

5.5.1 Spring Boot

Při práci s rámcem *Spring* je nutné nastavení *xml* konfiguračních souborů, řešící například datový zdroj, bez nichž aplikaci nelze spustit. *Spring Boot* je služba, která řeší velké množství nastavení aplikačního rámce *Spring* automaticky. Tato nastavení mají s použitím *Spring Boot* své vlastní výchozí hodnoty, které jsou koncipovány tak, že za použití jednoho z vestavěných

serverů (například *Tomcat* nebo *Jetty*) není potřeba konfigurací téměř žádná. Aplikaci je tak možné spustit velmi rychle a bez velké námahy.

Kód pro spuštění aplikace vypadá následovně:

```
@SpringBootApplication
public class GameService {
   public static void main(String[] args) {
      SpringApplication.run(GameService.class, args);
   }
}
```

Výpis 8: *Spring Boot* aplikace

Anotace *@SpringBootApplication* je zde nově představena. Z důvodu častého využití spojuje několik anotací dohromady:

- *@Configuration* (definuje třídu jako zdroj konfigurací),
- *@ComponentScan* (specifikuje, kde hledat komponenty a konfigurace)
- *@EnableAutoConfiguration* (odvozuje potřebné výchozí konfigurace dle definicí v *classpath*).

Výchozí hodnoty však pro provoz vlastních aplikací nemusí vždy stačit a přepsání těchto hodnot je vždy možné. Lze to uskutečnit vytvořením xml souboru, ačkoli se již tyto zdroje nedoporučuje používat[26]. Podporovanou konfigurací je ta, která se nachází přímo v kódu pomocí anotací *@Configuration*.

5.5.2 Rozšiřitelnost

Jak již bylo řečeno, rozšiřitelnost úkolů je prováděna na základě vytvoření nové služby. Tato služba může mít jakoukoli vnitřní strukturu, musí být ale schopna přijímat specifické typy zpráv.

- GET_AVAILABLE_QUEST_TYPES: Tento typ události musejí podporovat všechny služby, které chtějí, aby je bylo možné ve hře využít. Událost je vyvolána z prostředí webového editoru, kdy uživatel otevře okno pro vytvoření nového úkolu. Tím se systému zašle právě tato událost, společně se zpětnou "adresou", na kterou se mají data zaslat, a každá služba má na ni možnost reagovat. V odpovědi je nutno vytvořit novou událost, která musí být ve formátu QUEST_TYPE_[id]. *id* musí být pro každý typ úkolu unikátní. K této události je podstatné přiložit základní informace o úkolu nebo obecný příklad typu úkolu, které může editor využít jako vzor pro vytvoření úkolu vlastního.

  Jakou formu budou tato data mít záleží pouze na tom, jakou formou máme implementováno rozhraní korespondující s konkrétním typem úkolu. Strukturu těchto dat si tedy implementátor vytváří vlastní.

- SAVE/DELETE/UPDATE_QUEST_TYPE_[id]: Tyto události jsou vyvolány opět ve webovém editoru a slouží k jejich editaci.
- GET_ALL_QUEST_TYPE_[id]: Tato událost slouží pro podporu editace hry, díky níž nemusíme vytvářet stále objekty nové, ale můžeme vybírat z již existujících.
- GET_QUESTS_BY_IDS: Tato událost již slouží pro komunikaci s mobilní aplikací. Zobrazí-li si uživatel detail herního objektu, aplikace zjistí, jaké *ID* úkolů jsou k němu navázány a vyvolá zmíněnou událost. Odpověď na tuto událost musí obsahovat typ události QUEST_TYPE_[id]_BY_IDS, aby mohla aplikace správně určit, o jaký typ úkolu se jedná, a vyvolat třídy implementované právě pro něj.
- CHECK_SOLVED: Poslední nutnou zpracovatelnou událostí je událost CHECK_SOLVED. Je vytvořena mobilní aplikací, nese informace o počínání hráče při plnění úkolu, zajišťuje jejich zhodnocení a posílá zpět informaci o úspěchu či neúspěchu. Tato informace je událost typu IS_SOLVED s přiloženými daty o výsledku zhodnocení.

Kromě nároků na tyto typy úloh je zapotřebí mechanismu, který bude schopen připojení ke zprostředkovateli komunikace.

5.6 Komunikace

Implementace komunikační části je zvolena na základě již řečených skutečností, tedy tak, aby i v případě velké zátěže na systém dokázala doručovat zprávy stále efektivně. Zároveň bylo zapotřebí použít technologii zajišťující takovou formu komunikace, která by komunikační kanál odstínila od jakékoli jiné logiky kromě té, jak přeposlat zprávu z bodu A do bodu B. Nechtěla jsem tedy využít principy, jako je například *ESB*, na kterém by stála koordinace a řízení služeb, ale jednoduchý router, který by zprávy pouze doručoval.

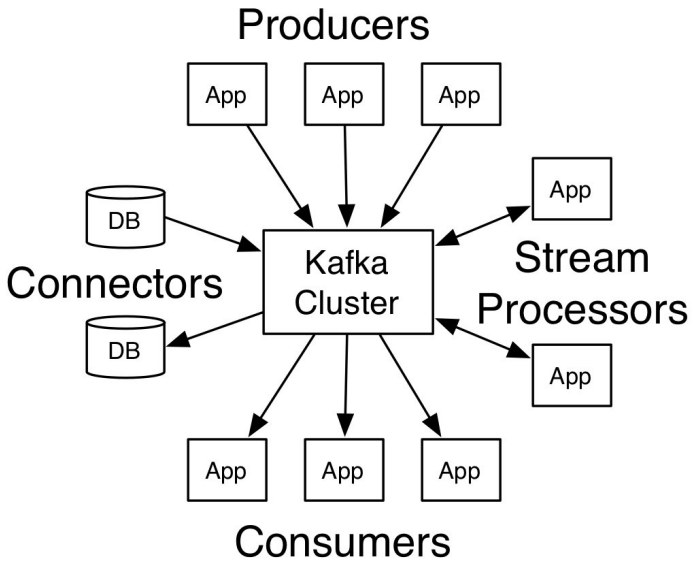
5.6.1 Apache Kafka

Kafka® is used for building real-time data pipelines and streaming apps. It is horizontally scalable, fault-tolerant, wicked fast, and runs in production in thousands of companies.

Apache Kafka je distribuovaná streamovací platforma pro manipulaci s velkými daty. Disponuje několika klíčovými vlastnostmi:

- Implementuje model *publish - subscribe* (vydavatel - odběratel)
- Ukládá proudy dat s vysokou tolerancí a odolností k chybám
- Zpracovává data v takové formě, v jaké se objeví

Kafka může běžet ve formě klastru na jednom nebo více serverech. Klastr ukládá data ve formě záznamů (record) do katerorií nazvaných témata (topics). Každý záznam se skládá z klíče, hodnoty a časového razítka.



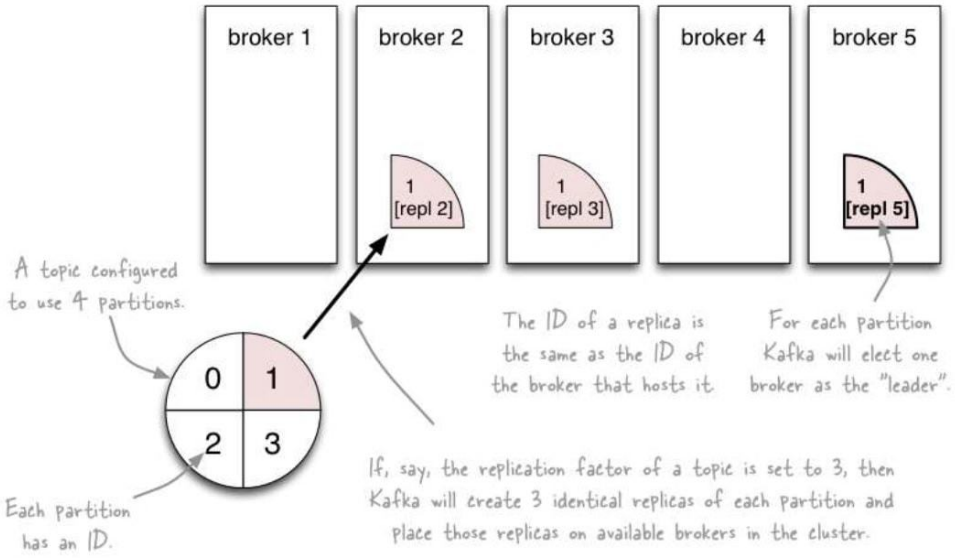
Obrázek 12: Princip platformy Apache Kafka

Kafka se v základu dělí do čtyř hlavních rozhraní:

- *Producer API*: Producentské rozhraní, které dovoluje aplikaci zasílat data do určených témat klastru
- *Consumer API*: Spotřebitelské rozhraní, které využívají aplikace k zapsání se k určitému tématu a přijímají tak záznamy, které do něj přes Producer API jiné aplikace vloží
- *Streams API*: Streamovací rozhraní nabízející rozhraní pro aplikaci tak, aby se mohla chovat jako stream procesor, přijímat záznamy z jednoho nebo více témat, efektivně je transformovat a obratem poslat do dalšího tématu nebo témat
- *Connector API*: Rozhraní pro konektory, které umožňuje sestavování a spouštění znovupoužitelných producentů a spotřebitelů pro existující aplikace

Grafické znázornění principu platformy *Apache Kafka* si lze prohlédnout na obrázku 12 na straně 45. Zasílání záznamů funguje na protokolu TCP.

Ke klastru se musí nejprve připojit producenti a konzumenti. Producent vytvoří záznam a vyšle jej na specifické téma, kde se záznam přiřadí jednomu z uzlů klastru, nazývaných broker. Škálovatelnost a distribuce systému *Kafka* je založena právě na možnosti velmi efektivního replikování těchto uzlů. V managementu distribuovaných částí hraje velkou roli další z produktů *Apache*, který nese jméno *Apache ZooKeeper*. *ZooKeeper* je centralizovaná služba pro správu konfigurací a zajišťuje synchronizaci distribuovaných a skupinových služeb[6]. Díky této službě se mohou uzly Kafka klastru rozšiřovat bez datových ztrát nebo desynchronizace.



Obrázek 13: Rozčlenění části tématu v clusteru

Po přijetí záznamu jedním z uzlů se záznam ukládá k příslušnému tématu. Při vysoké zátěži může být téma rozděleno do několik částí nazývaných partitions. Tyto části jsou neměnné sekvence záznamů, do kterých je kontinuálně zapisováno, a které jsou replikovány mezi několik uzlů. Každý záznam zde má svůj unikátní identifikátor. Témata, jejich rozdělení na části a replikace těchto částí je znázorněna na obrázku 13 na straně 46.

Členění samotných témat značně přispívá škálovatelnosti celého systému. Díky replikaci jednotlivých částí mezi uzly se pak zvyšuje tolerance k chybám, při ztrátě jednoho uzlu máme data zálohována v uzlech dalších. Všechny záznamy jsou v systému ukládány po určitou dobu, která může být nakonfigurována.

Spotřebitelé se mohou přihlašovat jak ke konkrétním tématům, tak i ke spotřebitelské skupině (consumer group). Službě *Kafka* se pak prokazují názvem skupiny. Patřit ke skupině znamená pro spotřebitele to, že se při vyprodukování určitého záznamu záznam nepošle všem konzumentům této skupiny, ale jen jednomu z nich, za účelem vyvažování zátěže v rámci skupiny. Pokud má každý konzument skupinu vlastní, chová se *Kafka* dle typického vzoru *publish - subscribe* a přeposílá záznamy dle daného tématu všem[8].

5.6.2 Připojení systémových komponent

Všechny služby, obsažené v geolokačním rámci, implementují nějakou formu konektoru, která je dokáže k systému *Kafka* připojit.

Každá služba se při vytvoření spojení zapíše k určitému tématu. Tato témata jsou definována typem událostí, které je služba schopna zpracovat. Tím se zajistí, že se vyvolání události z jakékoli části systému dostane na to správné místo.

5.7 WebSocket

WebSocket je jedním z komunikačních protokolů v počítačích, který poskytuje plně duplexní komunikační kanál. Tento kanál je při zahájení komunikace vytvořen pouze jedinkrát, spojení přes něj se až do ukončení přenášení kontinuálně udržuje.

Protokol, který *WebSocket* aplikuje, se skládá ze dvou částí. První částí je *handshake* (zahájení spojení), druhou částí je přenos dat. *Handshake* ze strany klienta je HTTP požadavek s žádostí o převod na jinou formu komunikace. Vypadá následovně:

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
```

Výpis 9: Hlavička *WebSocket*

Odpověď ze strany serveru je tato:

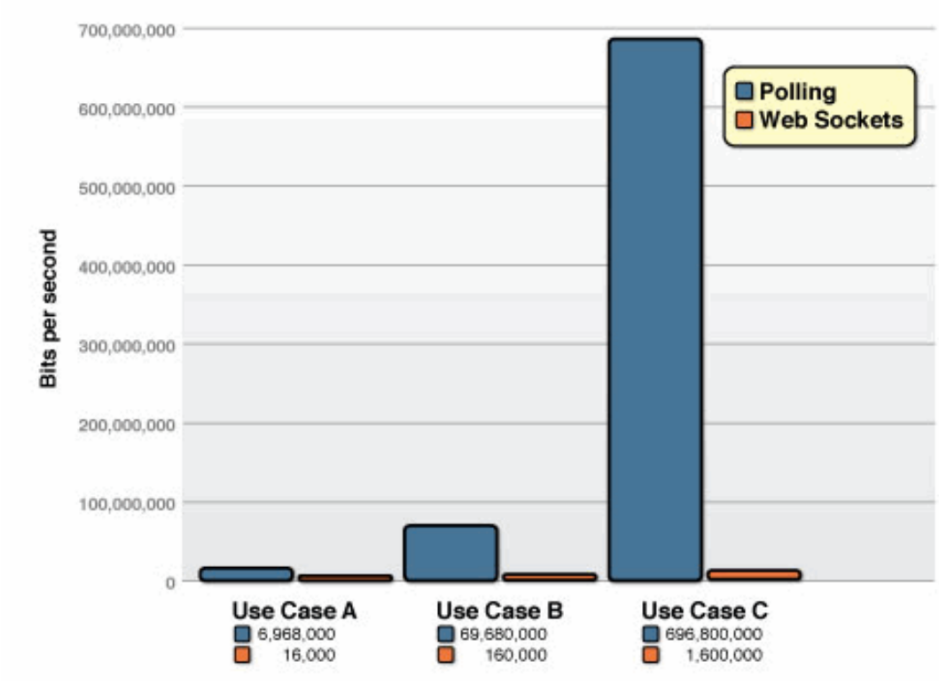
```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: chat
```

Výpis 10: Odpověď *WebSocket*

Momentem, kdy je navázání spojení úspěšné, mohou jak klient, tak server zasílat data druhé straně libovolně. Přenášení dat probíhá na základě TCP protokolu[3].

Využití komunikačního protokolu *WebSocket* je výhodné zejména při komunikaci systému a mobilní aplikace. Po prvotním navázání spojení si mohou obě strany, dle své vůle, zasílat zprávy bez toho, aniž by byla očekávána odpověď, což vede k nižší spotřebě dat, než při nutnosti zasílat potvrzující odpovědi. Mít neustále otevřený komunikační kanál také znamená, že při každém požadavku odpadá potřeba "obalovat"data velkým množstvím dodatečných informací a každý požadavek, co se týče těchto doplňujících skutečností, spotřebuje dat méně.

Na grafu na obrázku 14 na straně 48 můžeme vidět srovnání spotřeby dat protokolu *WebSocket* a protokolu HTTP [1]. S rostoucí zátěží uvedené v bitech za sekundu jsou zde znázorněny tři případy, které demonstrují rozdíly v množství přenášených dat při vysoké zátěži na systém. Uvažujeme-li situaci, kdy má HTTP hlavička dle příkladu 871 bajtů (převzato z [1]) a Socket hlavička bajty 2, dostaneme se k následujícím faktům:



Obrázek 14: Spotřeba dat komunikačních protokolů

- Use case A: Při zátěži 1000 klientů, kdy každý z nich obdrží jednu zprávu za sekundu, nejsou rozdíly mezi oběma protokoly velmi zřetelné.
- Use case B: Při zátěži 10000 klientů se podle množství 16000 bitů za sekundu protokol *WebSocket* jeví výrazněji jako úspornější řešení.
- Use case C: Při zátěži 100000 klientů, kteří obdržují jednu zprávu za sekundu, se HTTP protokol vůči *WebSocket* protokolu jeví jako velmi neefektivní.

Je nutno podotknout, že v mnoha případech může být použití protokolu HTTP výhodnější, než protokolu *WebSocket*. *WebSocket* protokol zasílá data jen pro udržení samotného spojení, tedy přenáší data i tehdy, kdy ani jedna ze stran nezaslala zprávu vědomě. Pro využití v tomto systému je však stěžejní zasílání asynchronních zpráv, které mohou udržovat stav hry aktuální v reálném čase a jejichž množství může být vysoké. Zvolila jsem proto řešení využívající protokolu *WebSocket*, konkrétně knihovny *kafka-websocket* vývojáře *b*[2].

5.8 Bezpečnost

Přihlašování uživatelů, jejich komunikace se systémem, rozdílné role a práva jednotlivých požadavků, ale i komunikace mezi komponentami systému vyžaduje určitou formu zabezpečení celého systému. Obyčejní hráči nesmějí mít povoleno provádět akce, které by narušily strukturu her a manipulovaly s objekty způsobem, který je možný jen pro editory her. Samotné komponenty

systému musí mít určena svá práva i práva komponent ostatních a při příchozích požadavcích kontrolovat, zda-li požadavek od dané komponenty s danými daty může být proveden.

S ohledem na architekturu systému a fakt, že je postavena na vzájemné komunikaci většího množství služeb, které jsem se snažila postavit tak, aby byly (pokud možno) co nejméně závislé na ostatních, hledala jsem technologii, která mé snažení nezhatí neustálým doptáváním a autentizováním každého příchozího požadavku na jednu komponentu u jiné, jakési autentizační komponenty. Zároveň jsem chtěla, aby vybraná technologie mohla být použita jak u služeb, tak i u klientů připojených z prohlížeče a z mobilního zařízení, aby nebylo nutné implementovat více rozhraní pro autentizaci a autorizaci požadavků v každé komponentě. Tohoto se mi povedlo dosáhnout využitím technologie *JSON Web Token*.

5.8.1 JSON Web Token

Ke zmíněné komunikaci uživatelů se systémem i komunikaci systémových komponent mezi sebou je použita technologie *JSON Web Token*, zkráceně *JWT*[7]. *JWT* je způsob bezpečné komunikace mezi dvěma entitami definován v RFC 7519. Jak již název napovídá, token se předává ve formátu JSON. Skládá se ze tří částí - hlavičky (*header*), přenášených dat (*payload*) a podpisu (*signature*). Tyto části jsou odděleny tečkou v pořadí "header.payload.signature".

Hlavička se obvykle skládá ze specifikace tokenu v "typ" klíči, který říká, že objekt je objektem JWT, a hešovacího algoritmu, za pomocí kterého se získává podpis.

```
{
   "typ": "JWT",
   "alg": "HS256"
}
```

Výpis 11: JWT Hlavička (*Header*)

V další části tokenu nalezneme informace o entitě, které token náleží. Datům obsaženým zde se říká požadavky (*claims*). Požadavků můžeme vytvořit, kolik chceme. Existují jisté standardy, např. "iss" klíč pro emitenta, "exp" pro expirační dobu tokenu nebo "sub" pro předmět, nejsou ale povinné. Při zasílání tokenu z jedné části systému na jiný jsou zasílány všechny tyto informace, příliš mnoho požadavků tak může způsobit zbytečně velký přenos dat a systém tak zpomalit.

```
{
    "sub": "00001",
    "name": "Jachym"
}
```

Výpis 12: JWT Data (*Payload*)

Hlavička a přenášená data jsou zakódována pomocí Base64 kódování. Společně s algoritmem specifikovaným v hlavičce a tajným klíčem je vytvořena poslední část tokenu, kterou je podpis. Poskládáním všech tří částí dohromady vytvoříme výsledný token, např:

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.
   eyJ1c2VySWQiOiJiMDhmODZhZi0zNWRhLTQ4ZjItOGZhYi1jZWYzOTA0NjYwYmQifQ.
      -xN\_h82PHVTCMA9vdoHrcZxH-x5mb11y1537t3rGzcM
```

Výpis 13: JWT token

Data přenášená pomocí *JWT* nejsou šifrována, slouží především pro potvrzení toho, že byla vytvořena důvěryhodným zdrojem. V této diplomové práci jsou tokeny použity hlavně z důvodu specifikace rolí dané entity.

5.8.2 Příklad použití

Uživatel se musí pro interakci se systémem přihlásit. Přihlášení probíhá ve službě *authentication service*. Služba převezme požadavek o přihlášení a v databázi zkontroluje, jestli je uživatel registrován a zadal správné přihlašovací údaje. Pokud ano, vytváří se token. V části tokenu payload se uloží všechny role, kterými daná entita disponuje, vygeneruje se výsledný token podle předchozího postupu a tento se zašle zpět uživateli. Na straně uživatele se token uloží do lokálního úložiště prohlížeče.

Ke každému dalšímu požadavku uživatele se přikládá uložený token. Každá služba v systému je schopna sama token přečíst a zjistit, jestli je validní, jaká jsou v něm data a hlavně, jakými rolemi uživatel disponuje. Na základě tohoto pak může filtrovat požadavky a akce, ke kterým má daný uživatel přístup.

6 Softwarové prostředí

Velké množství malých služeb je problémové spravovat. Nejen, že kvůli různým vývojovým prostředím vyvstávají různé problémy s vývojem jedné aplikace, problémy s vyvíjením mnoha malých aplikací se několikrát vynásobí. I když v prostředí systému Linux aplikace běží bez potíží, přenese-li se na systém Windows, vše může padnout. Nebo naopak. Lokalizace problému nebo několika problémů, které způsobily pád aplikace, může být v lepším případě jednoduchá, ale takovéto konfigurování mnoha malých služeb bude stále velmi náročné, minimálně časově.

Kromě problému integrace s prostředím je tu další háček. Máme-li mnoho malých aplikací, které chceme spustit, měli bychom využít službu, která nám je všechny sestaví a spustí automaticky. Zahájení běhu každé aplikace zvlášť se může stát velmi rychle strhujícím, obzvlášť, pokud běh některé služby závisí na službě jiné a jsou spuštěny ve špatném pořadí.

6.1 Virtuální stroj

Virtuální stroje jsou softwarovou technikou, která umožňuje vytvoření odděleného prostředí nezávislého na systému, ve kterém se nachází. V tomto odděleném prostředí mohou být nainstalovány systémy jiné. Máme-li tedy aplikaci, která je optimalizovaná pro systém Linux, ale v prostředí našeho PC je nainstalován systém Windows, řešením je spuštění virtuálního stroje na našem Windows PC, do kterého nainstalujeme systém Linux. Tento Linux systém neví, že běží v prostředí Windows a chová se stejně, jakoby běžel na jakémkoli jiném, nevirtuálním stroji. Zde můžeme spustit naši aplikaci a očekávat, že bude spuštěna úspěšně.

Problém nastává, když chceme spustit další aplikaci, také v systému Linux, ale s jinou verzí určitého softwaru, knihovny nebo aplikačního rámce, než využívá naše první aplikace. Spustíme-li ji na našem virtuálním stroji, start nebude úspěšný.

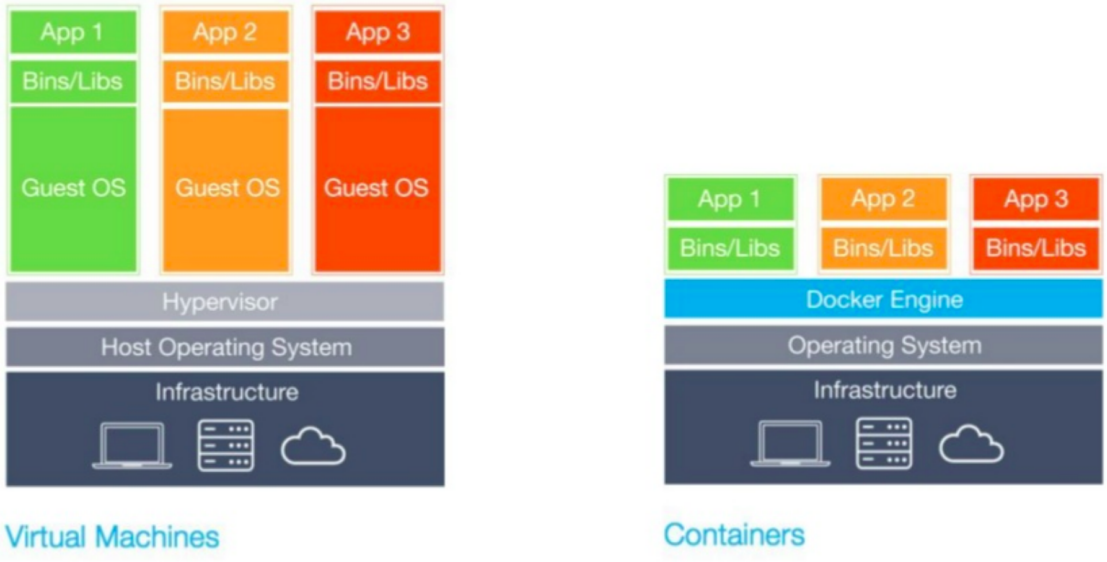
Řešení se nabízí ve formě vytvoření úplně nového virtuálního stroje, s novým systémem Linux, ale jinou verzí použitých závislostí. Toto řešení je funkční, ale náročné na zdroje. Každý virtuální zdroj si alokuje zdroje vlastní, které jsou často mezi virtuálními stroji duplicitní a od sebe navzájem izolované.

6.2 Docker

"Build, Ship, and Run Any App, Anywhere"[27]

Docker může vyřešit právě tyto nepříjemnosti. *Docker* staví na myšlence virtuálního stroje, ale posunuje ji na trochu jinou, efektivnější úroveň.

Protože obě Linuxová jádra mohou mít velké množství společných procesů, které si mezi sebou mohou dělit, aniž by se narušil chod aplikací, které v každém systému běží, prostředí *Docker* tyto procesy zaobaluje a vytváří základnu pro "další procesy". Tyto "další procesy" se nazývají *kontejnery*. *Kontejner* je izolovaná jednotka, která v sobě obsahuje vše, co potřebuje ke spuštění. V našem případě by v jednom kontejneru byla první aplikace s první verzí potřeb-



Obrázek 15: Porovnání virtuálního stroje a prostředí *Docker*

ného softwaru a ve druhém kontejneru aplikace s jeho druhou verzí. Výhodou kontejnerů je to, že nemusí obsahovat celou verzi operačního systému úplně od základu. Základ je poskytován prostředím *Docker*.

Grafické znázornění rozdílu mezi virtuálním strojem a prostředím *Docker* je ke shlédnutí na obrázku 15 na straně 52[28]. Na levé straně se nachází struktura virtuálních strojů. Můžeme vidět, že na hostitelském operačním systému je spuštěno několik systémů dalších. Na druhé straně lze vidět strukturu za použití prostředí *Docker*, kdy není potřeba spouštění operačních systému nových, všechny kontejnery využívají stejný *Docker Engine*.

6.2.1 Rychlost

Díky tomu, že kontejnery mohou využívat společnou část systému, která je již v provozu, mohou být spuštěny a připraveny v časovém rámci sekund, kdežto na nastartování celého virtuálního stroje by se čekalo i minuty.

6.2.2 Přenositelnost

Jeden a ten samý kontejner bude fungovat úplně stejně všude tam, kde je prostředí *Docker*. Nezáleží, jestli jeden kontejner replikujeme na jednom stroji nebo mezi více systémy, kontejner má v sobě všechen software, který ke svému provozu potřebuje, a ten si nosí všude s sebou.

6.2.3 Princip

Základem pro spuštění nového kontejneru je entita zvaná *image*. Tento *image* obsahuje celou naši aplikaci a vše, co ke svému provozu potřebuje. Proces vytvoření kontejneru je pak popsán

v souboru *Dockerfile*, nebo při složitějších operacích *docker-compose*. *docker-compose* může k provozu využít existující soubory *Dockerfile*. Využití technologie *Docker* v rámci této diplomové práce a důležité úkony pro spuštění celého systému je popsán v příloze *B*.

7 Nasazení

Systém spravován technologií *Docker* je možné jednoduše přenést kamkoli, kde je možné prostředí *Docker* vytvořit. Struktura založená na nezávislých kontejnerech je pak nasaditelná prakticky všude. V následujících řádcích tak uvedu stručný příklad, za použití jakých technologií by mohl být rámec nasazen.

7.1 Kubernetes

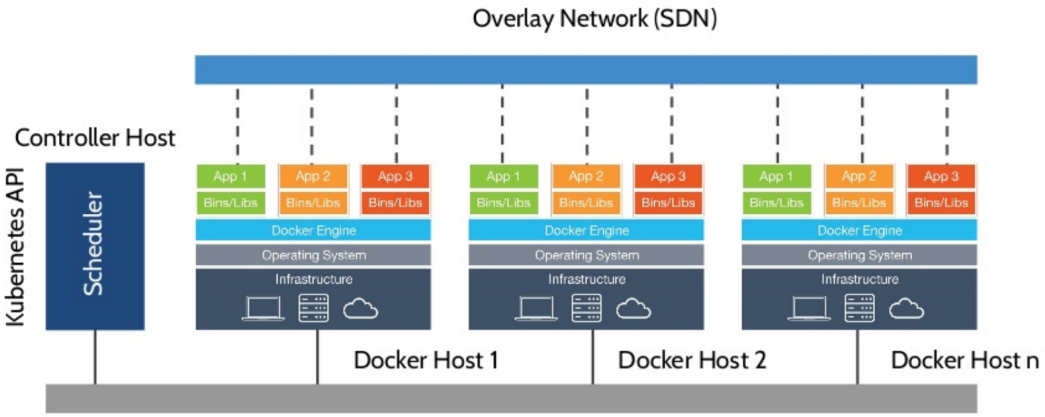
Kubernetes je příkladem technologie, která dokáže s technologií *Docker* efektivně pracovat. Je to systém, který umožňuje automatické nasazení, škálování a celkový management aplikací běžících v kontejnerech.[36] Vztah těchto dvou technologií je patrný z obrázku 16 na straně 55.

Nasazení geolokačního rámce do provozu by tedy mohlo být spravováno právě tímto orchestrátorem. S velkými možnostmi škálování, omezeními různých kontejnerů pro využití systémových zdrojů a jejich správou by se tak celý rámec zaštítil jednotnou centrální správou pro všechny komponenty.

7.2 Cloud

Aplikací, které využívají řešení *Cloud* v dnešní době stále přibývá. Je to z důvodu vysoké flexibility při škálování, hotová řešení zotavení při pádu aplikace, automatické aktualizace systému a mnoho dalších.[37]

Mezi nejrozšířenější cloudové služby se řadí bezpochyby *Google Cloud Platform* nebo *AWS*. Obě tyto služby nabízejí možnosti využití technologie *Kubernetes* pro správu kontejnerů, byly by tedy schopny vytvořit softwarové i hardwarové zázemí pro všechny vzniklé komponenty v geolokačním rámci.



Obrázek 16: Vztah technologie *Kubernetes* a *Docker*[38]

8 Zhodnocení rámce

Popsané technologie a principy, použité v architektuře geolokačního rámce, byly navrženy tak, aby zajišťovaly co nejoptimálnější výkon a přenášení dat mezi komponentami. Důvody jejich použití i výhody již byly popsány, takováto architektura však má i jisté stinné stránky.

Nepříjemnou skutečností navržené architektury bylo při vývoji i testování rámce to, že větší množství malých služeb, propojených komunikačním kanálem, samy o sobě, už jen ve svém klidovém stavu, vyžadovaly velké množství zdrojů. Stroj, na kterém byl vývoj prováděn, měl k dispozici 8 GB RAM a 4 GB odkládacího prostoru na disku. Při spuštění všech služeb najednou se dostal do stavu, kdy byla paměť i odkládací prostor téměř zaplněna.

To velmi znesnadňovalo vývoj způsobem, kdy bylo zapotřebí úprav v jedné službě, ovšem běh ostatních služeb byl stále vyžadován. Vedle všech běžících služeb muselo být tedy spuštěno i IDE, popřípadě webový prohlížeč pro možnost editace hry, což zatížilo systém ještě více, stroj byl na pokraji použitelnosti a komponenty nebyly schopné provozu.

Z hlediska výkonu má tak navržená architektura svá pro a proti. Oddělenými službami jsme získali vysokou modularitu, kdy každá služba může být implementována jiným způsobem, například v jiném jazyce, než ostatní, aby zajistila nejlepší a nejrychlejší výkon pro daný úkol, avšak každá z nich potřebuje pro svůj provoz vlastní zdroje. Přibyla také nutnost existence komunikačního kanálu mezi nimi, tedy zátěž na stroj byla navýšena nutností provozu komponenty pro odbavování komunikace. Získaná rychlost, flexibilita a modularita tak má za následek využití mnohem větší části zdrojů, než by tomu bylo u klasické, monolitické aplikace.

Z důvodu omezených zdrojů tak nebylo možné systém testovat jako celek. K otestování zátěže tak byl systém rozčleněn do několika částí, při kterých byly spuštěny jen ty služby, u kterých to bylo nezbytně nutné.

Kupříkladu, pro zhodnocení funkcionality systému při registraci a přihlášení většího množství uživatelů, bylo zapotřebí spustit službu *Geoframe-editor*, *Websocket-Kafka-Proxy*, *Kafka*, *Authentication-service* a *Authentication-db*. Prokázala se tak provozuschopnost daných částí z větším náporem uživatelů, ovšem určit provozuschopnost při běhu všech služeb rámce najednou pro mnoho uživatelů je otázkou výkonnějších zařízení, než jaké byly při stávající implementaci dostupné, popřípadě využít větší množství strojů.

8.1 Přenesená data a zatížení mobilního zařízení

Zaměřením na to, aby byl uživatel mobilní aplikace co možná nejvíce odstíněn od přebytečného náporu spotřeby dat, se ukázalo být správným krokem vytvoření *Localization-service*. Opět tedy, abychom něco získali, museli jsme něco ztratit. Provoz další služby je opět konzumentem nových zdrojů. Díky tomu jsme však získali prostředníka mezi klientskou mobilní aplikací a službou *Game-objects-service*, který je schopen filtrovat objekty, které již byly uživateli zaslány, od objektů, které je teprve třeba zaslat.

Obrázek 17: Rámec v využitím *Localization-service*

Při porovnání s ostatními myšlenkami řešení filtrace objektů, *Localization-service* pomohla k redukci přenášených dat uživateli nejvíce. Způsob, kdy by se filtrace objektů neprováděla vůbec, je možný při malém množství herních entit pro krátké hry na malém území, ovšem při velkém objemu objektů je tento přístup nemyslitelný. Vyřešil by sice problém, jak uživateli zaslat objekty, které byly po čase jejich načtení v aplikaci pozměněny editorem, ale za velkou cenu duplicitních dat jiných objektů u koncového uživatele.

Přijatelnějším řešením by bylo, při oznámení polohy od uživatele směřující systému, přibalit k notifikaci data, která by nesla identifikátory všech objektů, které již uživatel obdržel. Tak by se filtrace přenesla na službu *Game-objects-service* a v celkovém systému by oproti využití *Localization-service* opadlo přenášení dat objektů ze služby na službu. Ze strany systému je tato možnost výhodnější, protože zajišťuje nižší objem přenášených dat mezi službami. Klientská mobilní aplikace je však takto v nevýhodě, protože musí zasílat informace o již načtených objektech.

Stávající řešení s využitím *Localization-service* sráží data, přenášená mezi mobilní aplikací a zbytkem systému, na minimum. Může se tak stát "mezipamětí" i pro různé typy úkolů, které by pro svůj provoz využívaly stejná data. S novým úkolem by se tak dokázaly filtrovat informace, které již uživatel obdržel, a v aplikaci poté tyto informace sdílet.

Dalším prvkem, který šetří přenášení dat k uživateli, je postupné načítání informací o objektech. Připojením mobilní aplikace se přenesou pouze základní data o objektech, jako je název, popis a oblast, ze které může být k objektu přistoupeno, popřípadě úkoly, které souvisí se vzdáleností hráče od objektu. Detailní informace se načtou až na vyžádání. V souvislosti s tímto se osvědčilo využití komunikačního protokolu *WebSocket*, který udržuje stálé spojení a oproti protokolům typu "request-response"dokáže data zasílat volně, bez nutnosti "obalování"informací dodatečnými daty.

Na grafech 17 na straně 57 a 18 na straně 58 můžeme vidět rozdíl běhu rámce při použití přístupu využívajícího *Localization-service* a přístupu bez něj. Oba grafy zachycují bezprostřední start aplikace, přihlášení a připojení se k dané hře. Po načtení objektů byl jeden objekt vybrán,

Obrázek 18: Rámec bez využití *Localization-service*

vyřešen úkol k němu se vztahující a zobrazen bod nový. Pro test byla využita hra, při které se načítalo pět objektů se svými základními vlastnostmi se synchronizací nastavenou na dvě sekundy. Z grafů *Network* lze vysledovat, že přítomnost *Localization-service* zdárně odstínila nutnost zasílání již načtených objektů a využití dat se tak omezilo jen na přihlášení a prvotní vykreslení mapy. Další využití dat, patrné z grafu, mělo za následek načtení detailu objektu s úkolem, který byl vyřešen, a nového objektu, který byl tímto získán.

Vzhledem k tomu, že stávající implementace neobsahuje velké množství grafických prvků, jako je to např. u zmiňovaných aplikací *Ingress* nebo *Pokemon Go*, zátěž na baterii, u těchto aplikací tak výrazná, zde není neobvykle vyčerpávající. Vykreslování uživatelského rozhraní se týká hlavně map a herních objektů. Nevyhnutelná zátěž pak pochází primárně z nutnosti mít zapnutou GPS lokalizaci a z přenášení dat.

Technika, která je při lokalizaci a načítání nových objektů použita, odstiňuje přebytečnou komunikaci aplikace a zbytku systému. Díky dostupné lokalizaci polohy uživatele knihovny *osmdroid* aplikace určuje, jestli se uživatel pohnul dostatečně daleko na to, aby bylo zažádáno o nové vykreslení okolních skutečností. Uložením polohy, ve které byl uživatel naposledy při žádosti o data, je možné zjistit, jakou vzdálenost hráč od té doby urazil. Podle toho, jak velkou má hra nastavenou oblast pro vykreslování objektů, je určena hranice, za kterou je už nutné nové objekty vykreslovat. Aktuální implementace počítá s poloviční vzdáleností poloměru oblasti pro vykreslování každé hry.

9 Závěr

Cílem této práce bylo vytvoření rámce pro geolokační hry, který by umožnil vytváření různých verzí her založených na poloze hráče a jeho navigace k zájmovým bodům hry. Vytvoření tohoto rámce se skládalo z několika částí.

První částí byl průzkum aktuálního stavu geolokačních her. Prozkoumáním různých her, založených na různých principech hraní, byly vymezeny tři základní verze, do kterých je možné hry rozdělit. Byly to hry založené pouze na souřadnicovém systému GPS, hry, které, kromě GPS, využívaly i podpůrné mobilní aplikace pro navigaci a zobrazování informací o herních skutečnostech, a hry, které podporovaly mód s více hráči v reálném čase.

Tyto skutečnosti se staly základem pro návrh architektury, která je koncipována tak, aby zvládla jak navádění jednoho hráče hrou, tak měla prostředky pro provoz hry společné pro více hráčů. Využitím principů architektury orientované na služby, nezávislé na ostatních, se povedlo vytvořit modulární systém, který je schopen chovat se různými způsoby, v závislosti na tom, jakých služeb je využito.

Tato skutečnost vyřešila další cíl této práce, kterým byla rozšiřitelnost jednotlivých typů herních úkolů. Přidání logiky pro daný typ úkolu probíhá formou připojení zcela nové služby. Tak se rozšíření o nové typy stalo zcela nezávislé na implementaci již stávajících komponent. V kontextu uživatelského rozhraní je samozřejmě také potřeba přidání nových komponent pro zobrazovaní akcí souvisejících s úkolem. To je zajištěno nabízeným rozhraním, které musí specifická okna implementovat, aby dokázala se zbytkem aplikace komunikovat.

Dalším bodem k prozkoumání byla možnost využití *offline* režimu při práci mobilní aplikace. Tato funkcionalita se prokázala být komplikovanou. Problémem je orchestrace všech komponent a různorodých úkolů tak, aby bylo jednoznačně určeno, jak s každou komponentou nakládat a jakými kroky lze dospět do stavu, kdy by byly všechny herní možnosti přístupné i bez připojení k internetu. To vše se zachováním modularity, nízké propojenosti a izolace všech služeb. Jednoznačné řešení v této diplomové práci tak není specifikováno, je však navrhnuto několik možných způsobů provedení a zanalyzována pro a proti pro každé.

Cílem implementace bylo umožnění vytváření a editace geolokačních her. Tento cíl naplnila existence webového editoru, ve kterém může uživatel vytvořit prostředky k založení nových her a s nimi spjatých herních objektů na jím vybraných místech. Správa uživatelů a her, která je dalším požadavkem na funkcionalitu rámce, je zajištěna přidělením rolí uživatelům a omezením jejich pravomocí. Hry tak mají určeného uživatele, který má právo s nimi manipulovat a určovat jejich vlastnosti. Tyto vlastnosti může mít každá hra jiné, čímž se zajistila možnost vytváření různých verzí her. Posledním požadavkem na implementaci byla rozšiřitelnost jednotlivých typů úkolů, která je zajištěna již zmíněnou architekturou systému.

Závěrečné zhodnocení rámce prokázalo, že problematické oblasti, jako je například velké množství přenášených dat nebo jejich synchronizace, je možné takto navrženým rámcem vyřešit. Projevila se však nevýhoda tohoto systému, a tou je náročnost na zdroje. Jejich nedostatkem

nemohl být systém otestován ve svém plném rozsahu, nedá se tedy jednoznačně určit, jaký nápor současně připojených uživatelů zvládne.

Z uvedených skutečností se, dle mého názoru, všechny hlavní cíle projektu podařilo splnit. Implementace dle popsaného návrhu obsahuje základní mechanismy, které slouží jako ukázka pro budoucí vývoj mnoha rozličných herních akcí a projektují, jakým způsobem je možné tyto akce do systému zakomponovat. Uživatelská rozhraní umožňují jak vytváření nových her, tak jejich provoz a jsou uzpůsobeny k tomu, aby byly v případě nových akcí rozšiřitelné nebo zaměnitelné. Fantazii dalšího vývoje systému se tak meze nekladou. Ať už je to rozšíření grafických či logických prvků, rámec je na to připraven.

Literatura

[1] LUBBERS Peter, GRECO Frank. *HTML5 WebSocket: A Quantum Leap in Scalability for the Web* [online] [cit. 2018-02-10] Dostupné z: http://websocket.org/quantum.html

[2] BLACK Benjamin. *kafka-websocket* [online] [cit. 2018-02-10] Dostupné z: https://github.com/b/kafka-websocket

[3] FETTE Ian., MELNIKOV Alexey. *The WebSocket Protocol* [online] [cit. 2018-02-20] https://tools.ietf.org/html/rfc6455#section-1.1 - websockety

[4] GAMMA Erich, HELM Richard, JOHNSON Ralph, VLISSIDES John (Gang of Four): *Návrh programů pomocí* vzorů. Grada. Praha 2003. ISBN 8024703025

[5] WICKRAMARACHCHI Anuradha. *Event Driven Architectrute Pattern* [online] [cit. 2018-04-15] Dostupné z: https://towardsdatascience.com/event-driven-architecture-pattern-b54fc50276cd

[6] *The Apache Software Foundation. Apache ZooKeeper* [online] [cit. 2018-01-10] Dostupné z: https://zookeeper.apache.org/

[7] JONES B. Michael, BRADLEY John, SAKIMURA Nat. *JSON Web Token (JWT)* [online] [cit. 2018-04-10] Dostupné z: https://tools.ietf.org/html/rfc7519

[8] *codeflex.co. What is Apache Kafka* [online] [cit. 2018-04-15] Dostupné z: http://codeflex.co/what-is-apache-kafka/

[9] FOWLER Martin. *Microservices* [online] [cit. 2018-04-16] Dostupné z: https://martinfowler.com/articles/microservices.html

[10] FOWLER Martin. *ServiceOrientedAmbiguity* [online] [cit. 2018-04-16] Dostupné z: https://martinfowler.com/bliki/ServiceOrientedAmbiguity.html

[11] HUŇKA, František. *Konstrukční návrhové vzory* [online] [cit. 2018-04-16] Dostupné z: http://www1.osu.cz/ hunka/vyuka/javaOOP/objap/objap_8/APNVZ_07x.pdf

[12] RYCHLÝ Marek. *SERVISNĚ ORIENTOVANÁ ARCHITEKTURA A JEJÍ APLIKACE V SYSTÉMECH SLEDOVÁNÍ A ŘÍZENÍ VÝROBY* [online] [cit. 2018-04-20] Dostupné z: http://www.fit.vutbr.cz/ rychly/public/docs/arap11.soa-a-jeji-apl-pri-rizeni-vyroby/arap11.soa-a-jeji-apl-pri-rizeni-vyroby.pdf

[13] Lidová kultura. (2007). Národopisná encyklopedie Čech, Moravy a Slezska. 2.svazek. Praha: Mladá fronta.

[14] Empirica spol., TÜV Rheinland spol. *Mobile BroadbandPrices in Europe 2017* [online] [cit. 2018-04-20] Dostupné z: https://ec.europa.eu/digital-single-market/en/news/mobile-broadband-prices-europe-2017 studie eu o cenach dat (2017)

[15] CC Attribution-Noncommercial 4.0 International. *Geocaching Live* [online] [cit. 2018-04-16] Dostupné z: https://www.geoget.cz/doku.php/user:gclive

[16] *Groundspeak, Inc.*, *Go geocaching anytime, anywhere.* [online] [cit. 2018-04-20] Dostupné z: https://www.geocaching.com/play/mobile

[17] *Official U.S. government information. What is GPS?* [online] [cit. 2018-03-15] Dostupné z: https://www.gps.gov/systems/gps/

[18] KETTLER, Audrey. *3 million geocaches: the infographic.* [online] [cit. 2018-03-15] Dostupné z: https://www.geocaching.com/blog/2017/04/3-million-geocaches-the-infographic/

[19] *Oficiální webové stránky hry. KÓD SALOMON*[online] [cit. 2018-04-20] Dostupné z: https://kodsalomon.ostrava.cz/

[20] *GIT repozitář knihovny.. osmdroid* [online] [cit. 2018-04-20] Dostupné z: https://github.com/osmdroid/osmdroid

[21] *Oficiální webové stránky. Google Maps JavaScript API Usage Limits* [online] [cit. 2018-04-20] Dostupné z: https://developers.google.com/maps/documentation/javascript/usage

[22] *Agile Store Locator. Google Maps vs Open Street Maps Comparison* [online] [cit. 2018-04-20] Dostupné z: https://agilestorelocator.com/blog/google-maps-vs-open-street-maps-comparison/

[23] OpenStreetMap Wiki. *Tiles* [online] [cit. 2018-04-20] Dostupné z: https://wiki.openstreetmap.org/wiki/Tiles

[24] Oficiální webové stránky. *AngularJS* [online] [cit. 2018-04-20] Dostupné z: https://angularjs.org/

[25] RUBERT, David. *Documentation, angular-leaflet-directive* [online] [cit. 2018-04-20] Dostupné z: http://tombatossals.github.io/angular-leaflet-directive/#!/

[26] Oficiální dokumentace rámce Spring. *Spring Boot Reference Guide* [online] [cit. 2018-04-20] Dostupné z: https://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/htmlsingle/#boot-features-external-config-validation

[27] Oficiální webové stránky. *Docker* [online] [cit. 2018-04-20] Dostupné z: https://www.docker.com/

[28] Docker, Inc. *Docker Birthday #3 - Intro to Docker Slides* [online] [cit. 2018-04-20] Dostupné z: https://www.slideshare.net/Docker/docker-birthday-3-intro-to-docker-slides

[29] Soa Vs Microservices Awesome Microservices Vs soa Gseokbinder [online] [cit. 2018-04-20] Dostupné z: http://architecture-nice.com/soa-vs-microservices/soa-vs-microservices-awesome-microservices-vs-soa-gseokbinder/

[30] *Singletons vs. Application Context in Android?*[online] [cit. 2018-04-20] Dostupné z: https://stackoverflow.com/questions/3826905/singletons-vs-application-context-in-android

[31] KERIEVSKY Joshua. *Refactoring to Patterns*. Addison-Wesley, 2005. ISBN 0321213351.

[32] WHITE, Jim. *Android's Application Class* [online] [cit. 2018-04-20] Dostupné z: https://www.intertech.com/Blog/androids-application-class/

[33] Oficiální dokumentace Android developers. *Activity*. [online] [cit. 2018-04-20] Dostupné z: https://developer.android.com/reference/android/app/Activity.html

[34] FOWLER, Martin. *What do you mean by "Event-Driven"?* [online] [cit. 2018-04-20] Dostupné z: https://martinfowler.com/articles/201701-event-driven.html

[35] *Everything Should Be Made as Simple as Possible, But Not Simpler* [online] [cit. 2018-04-20] Dostupné z: https://quoteinvestigator.com/2011/05/13/einstein-simple/

[36] Oficiální webové stránky. *Kubernetes* [online] [cit. 2018-04-20] Dostupné z: https://kubernetes.io/

[37] SALESFORCE UK. *Why Move To The Cloud? 10 Benefits Of Cloud Computing* [online] [cit. 2018-04-20] Dostupné z: https://www.salesforce.com/uk/blog/2015/11/why-move-to-the-cloud-10-benefits-of-cloud-computing.html

[38] GUNARANTNE Imesh. *Revolutionizing WSO2 PaaS with Kubernetes & App Factory* [online] [cit. 2018-04-20] Dostupné z: https://www.slideshare.net/imesh/revolutionizing-wso2-paas-with-kubernetes-app-factory

A Příloha na CD/DVD

Tato elektronická příloha obsahuje zdrojové kódy implementovaného rámce a text přílohy *B*, který uvádí postup, jakým lze rámec uvést do provozu.

B Spuštění rámce a jeho funkcionalita

B.1 Sestavení a spuštění

Nejjednodušší sestavení a spuštění rámce je vázáno na platformu *Docker*. V kořenové složce v příloze *A* se nachází soubor *docker-compose.yml*. Provedením příkazu *docker-compose up* se automaticky vytvoří *image* pro technologii *Kafka* (*spotify/kafka*), webový prohlížeč (*geoframe-editor*), proxy službu pro propojení uživatelských aplikací a zbytku systému (*geoframe-ws-kafka*) a všech databází, které jsou službami využívány.

Image pro služby ve složce *geoframe-services* takto sestaven není. Jeho vytvoření je závislé prvně na sestavení projektu pomocí systému *maven*, konkrétně pluginu *dockerfile-maven-plugin*. Před spuštěním příkazu *docker-compose up* je tak, pro každou službu ve složce *geoframe-services*, nutno provést příkaz *mvn install*, který vytvoří *image* pro konkrétní službu. Až poté je možné provést příkaz *docker-compose up*, který vytvoří *image* ostatních komponent a vytvoří a spustí pro každou z nich vlastní kontejner.

Po spuštění všech kontejnerů je možné připojit se k webovému editoru běžícímu na portu 8000. Porty lze změnit v souboru *docker-compose.yml*.

Pro propojení aplikace se systémem je nutno znát IP adresu zařízení, na kterém je systém spuštěn.

B.2 Funkcionalita

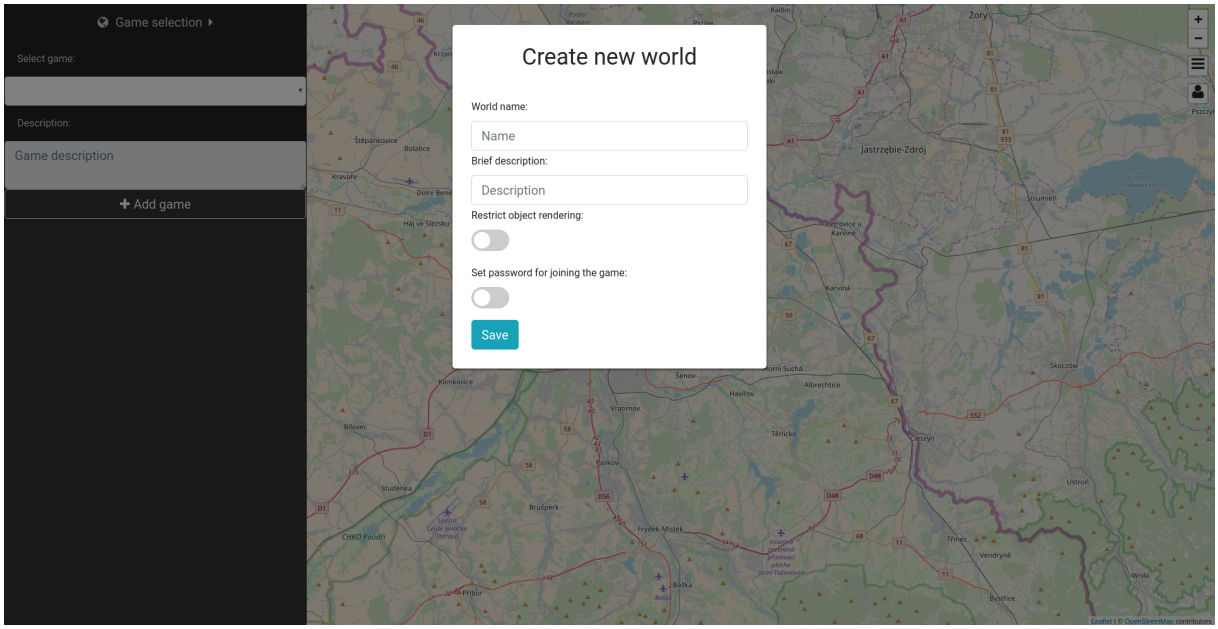
B.2.1 Webový prohlížeč

Ve webovém prohlížeči je nutno se nejprve přihlásit nebo registrovat. Tím se otevře možnost přidání nové hry, znázorněna na obrázku 19 na straně 66. Zde může uživatel zadat název, popis nebo příběh hry a oblast, která bude sloužit pro určení vzdálenosti generování nových objektů v mobilní aplikaci.

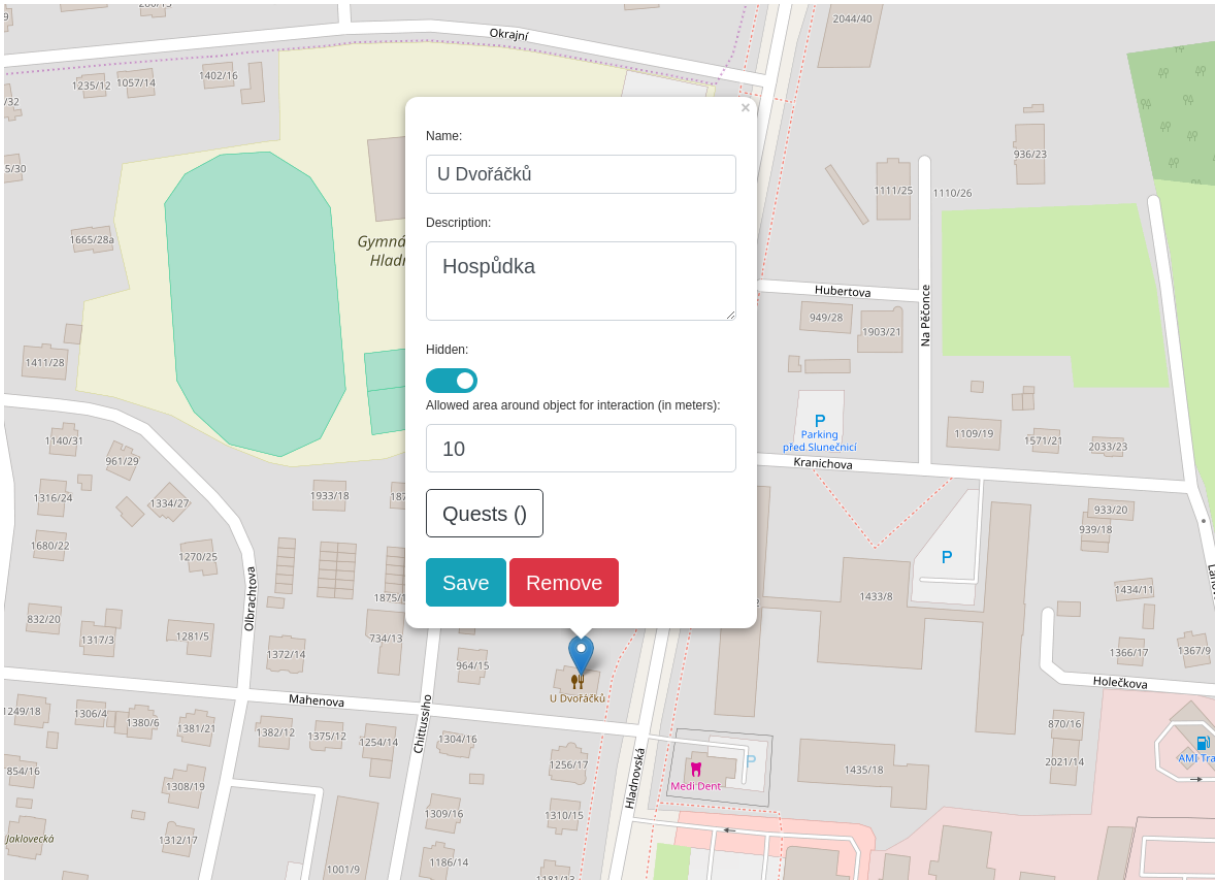
Po vytvoření hry může hráč po kliknutí na určité místo na mapě přidávat nové objekty. Otevře se nové okno, ukázané na obrázku 20 na straně 66. Na tomto obrázku probíhá vytvoření objektu *U Dvořáčků*, který je skrytý a je možné s ním interagovat ve vzdálenosti nejvíce 10 metrů.

Následně může uživatel k objektu připojit úkol prostřednictvím tlačítka *Quests*. Na obrázku 21 na straně 67 je znázorněno vytvoření nového úkolu *Zvirata* s otázkou směřovanou k hráči, na kterou musí odpovědět, aby dokázal postoupit ve hře dál.

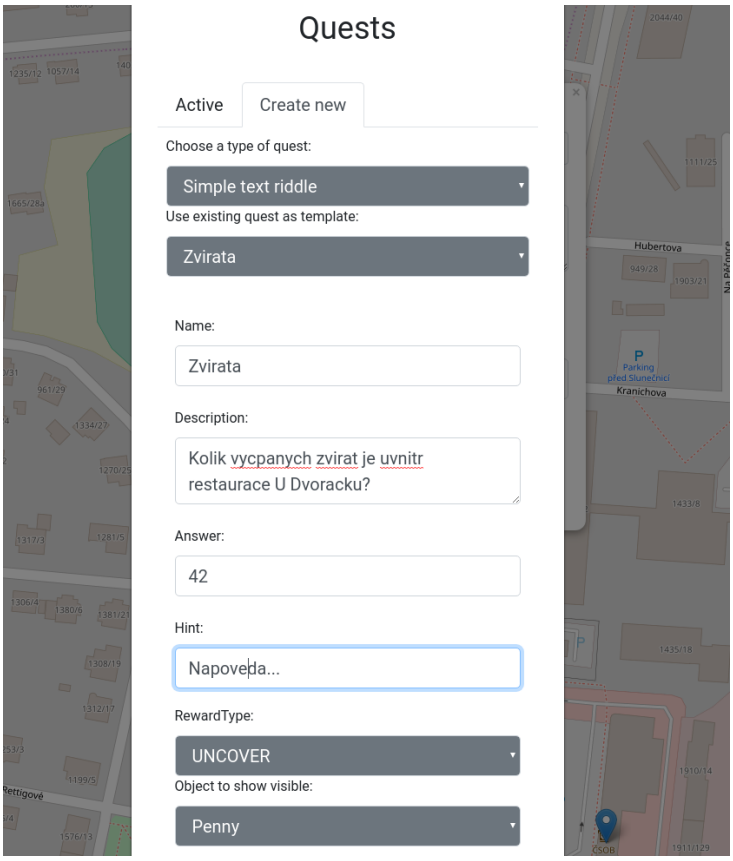
Zhodnocení aktivních úkolů probíhá také prostřednictvím tlačítka *Quests* v záložce *Active* znázorněné na obrázku 22 na straně 67



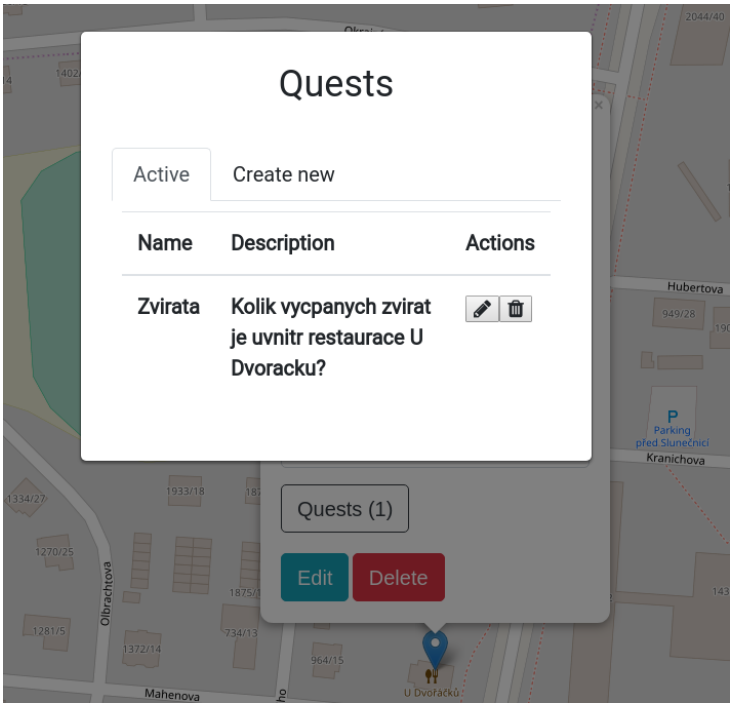
Obrázek 19: Vytvoření nové hry



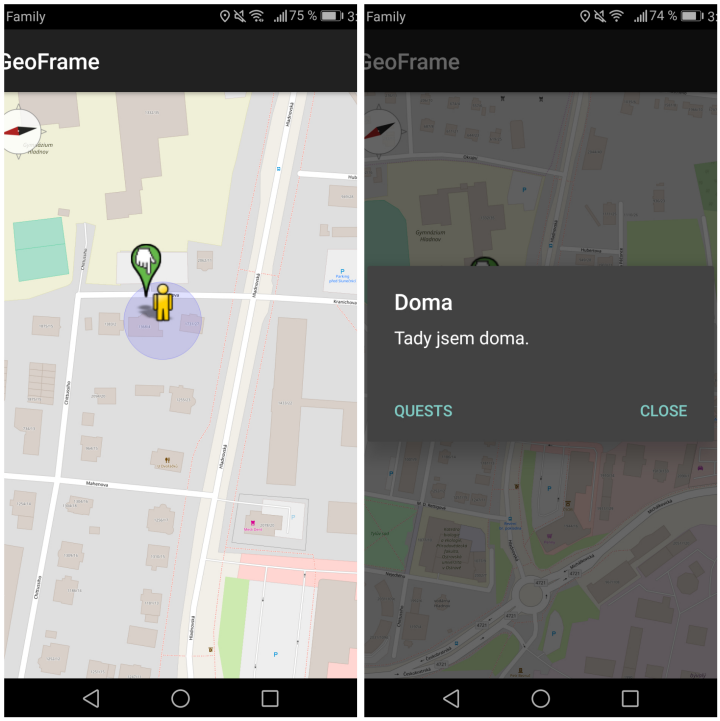
Obrázek 20: Vytvoření nového objektu



Obrázek 21: Vytvoření nového úkolu



Obrázek 22: Zobrazení aktivních úkolů



Obrázek 23: Začátek hry

B.2.2 Mobilní aplikace

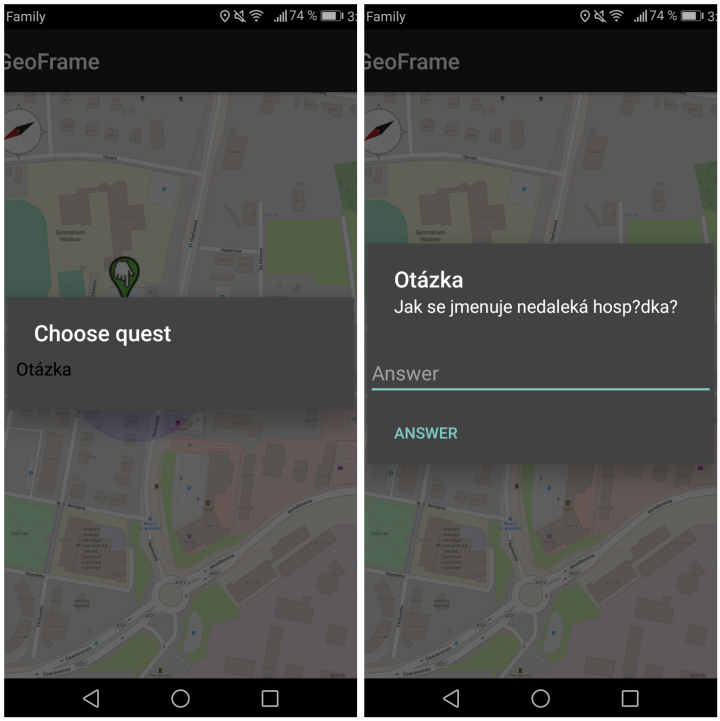
V mobilní aplikaci je nutno se přihlásit. Po přihlášení uživatel vybere hru, které se chce zúčastnit a výběr potvrdí. Hra na obrázku 23 na straně 68 začíná jediným objektem v uživatelově blízkosti. Po kliknutí na tento objekt se zobrazí detailní informace o něm. Na obrázku 24 na straně 69) pak vidíme dialog s úkoly, které jsou na objekt navázány. Po správném zodpovězení otázky na daný úkol se odkryje nový objekt, konkrétně *U Dvořáčků* (obrázek 25 na straně 69), který byl vytvořen v předchozí kapitole při editaci.

B.2.3 Uživatelé

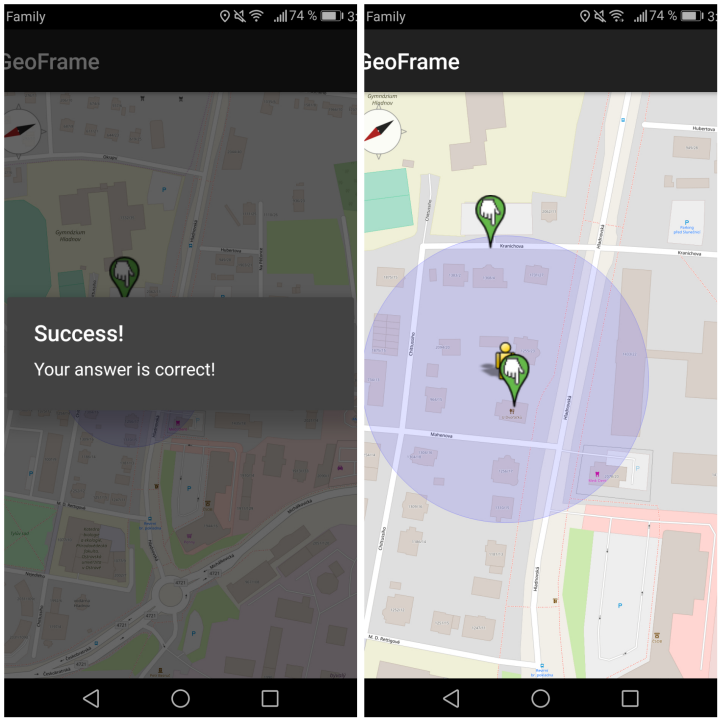
V současnosti rámec podporuje tři uživatelské role:

- ROLE_USER
- ROLE_OWNER
- ROLE_PLAYER

Spuštěním rámce se automaticky vytvoří uživatelé *marvin1-5* (*marvin1, marvin2, marvin3...*) s heslem *123* a rolí *ROLE_USER*.



Obrázek 24: Zobrazení úkolu



Obrázek 25: Zhodnocení úkolu a odkrytí nového objektu